

## 前言

### Mybatis入门

1. 什么是MyBatis
2. 为什么我们要用Mybatis?
3. Mybatis快速入门
  - 3.1 导入开发包
  - 3.2 准备测试工作
  - 3.3 创建mybatis配置文件
  - 3.4 编写工具类测试是否获取到连接
  - 3.5 创建实体与映射关系文件
  - 3.6 编写DAO
4. Mybatis工作流程
5. 完成CRUD操作
  - 5.1 增加学生
  - 5.2根据ID查询数据
  - 5.3 查询所有数据
  - 5.4 根据id删除
  - 5.5 修改
  - 5.6 小细节
  - 5.7Mybatis分页
6. 动态SQL
  - 6.1 动态查询
  - 6.2 动态更新
  - 6.3 动态删除
  - 6.4 动态插入
7. 入门总结

### Mybatis配置信息

1. 映射文件
  - 1.1 占位符
  - 1.2 主键生成策略
    - 1.2.1 UUID
  - 1.3 主键返回
  - 1.4 resultMap
  - 1.5 resultMap和resultType区别
  - 1.6 使用resultMap
  - 1.7 resultType和resultMap用法总结
  - 1.8Mybatis映射文件处理特殊字符
2. 配置文件
  - 2.1别名
  - 2.2Mapper加载
  - 2.3延迟加载
  - 2.4 延迟加载测试
3. 配置相关总结

### 关联映射

1. Mybatis 【多表连接】
  - 1.1一对一
    - 1.1.1设计表:

- 1.1.2 实体
- 1.1.3 映射文件
- 1.1.4 DAO层
- 1.2 一对多
  - 1.2.1 设计数据库表
  - 1.2.2 实体
  - 1.2.3 映射文件SQL语句
  - 1.2.4 DAO
- 1.3 多对多
  - 1.3.1 数据库表
  - 1.3.2 实体
  - 1.3.3 映射文件
  - 1.3.4 DAO

## 2. 关联映射总结

## 缓存+Mapper代理+逆向工程

- 1. 前言
- 2. Mybatis缓存
  - 2.1 Mybatis一级缓存
  - 2.2 Mybatis二级缓存
  - 2.3 Mybatis二级缓存配置
  - 2.4 查询结果映射的pojo序列化
  - 2.5 禁用二级缓存
  - 2.6 刷新缓存
  - 2.7 了解Mybatis缓存的一些参数
- 3. mybatis和ehcache缓存框架整合
  - 3.1 整合jar包
  - 3.2 ehcache.xml配置信息
  - 3.3 应用场景与局限性
    - 3.3.1 应用场景
    - 3.3.2 局限性
- 4. Mapper代理方式
  - 4.1 Mapper开发规范
  - 4.2 Mapper代理返回值问题
- 5. Mybatis解决JDBC编程的问题
- 6. Mybatis逆向工程
  - 6.1 修改pom.xml文件
  - 6.2 generatorConfig.xml配置文件
  - 6.3 使用插件步骤
  - 6.4 最后生成代码
- 7. 本章总结

## Mybatis整合Spring

- 1. Mybatis与Spring整合
  - 1.1 导入jar包
  - 1.2 创建表
  - 1.3 创建实体
  - 1.4 创建实体与表的映射文件
  - 1.5 创建Mybatis映射文件配置环境
  - 1.6 配置Spring核心过滤器【也是加载总配置文件】
  - 1.7 配置数据库信息、事务

1.8 创建Dao、Service、Action

1.9JSP页面测试

2. 总结

### Mybatis常见面试题

1. `#{}和${}` 的区别是什么？
2. 当实体类中的属性名和表中的字段名不一样，怎么办？
3. 如何获取自动生成的(主)键值？
4. 在mapper中如何传递多个参数？
5. Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？
6. Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？
7. 为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？
8. 通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？
9. Mybatis比IBatis比较大的几个改进是什么
10. 接口绑定有几种实现方式,分别是怎么实现的？
11. Mybatis是如何进行分页的？分页插件的原理是什么？
12. 简述Mybatis的插件运行原理，以及如何编写一个插件
13. Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？
14. Mybatis都有哪些Executor执行器？它们之间的区别是什么？
15. MyBatis与Hibernate有哪些不同？

## 前言

---

这个文档的内容纯手打，如果想要看更多的干货文章，关注我的公众号：**Java3y**。有更多的原创技术文章和干货！

目前疯狂处于疯狂更新PDF中，只要是Java后端的知识，都会有！欢迎来我公众号催更！微信搜索：**Java3y**

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！公众号有我的联系方式



- 🔥Java精美脑图
- 🔥Java学习路线
- 🔥开发常用工具
- 🔥精美原创电子书

在公众号下回复「888」即可获得！！

学习不能盲目，跟着我，会让你事半功倍

文档允许随意传播，但不能修改任何内容。

电子书的整理也是挺不容易，如果你觉得有帮助，想要打赏作者，那么可以通过这个收款码打赏我，金额不重要，心意最重要。主要是我可以通过这个打赏情况来预计大家对这本电子书的评价，嘻嘻



# Mybatis入门

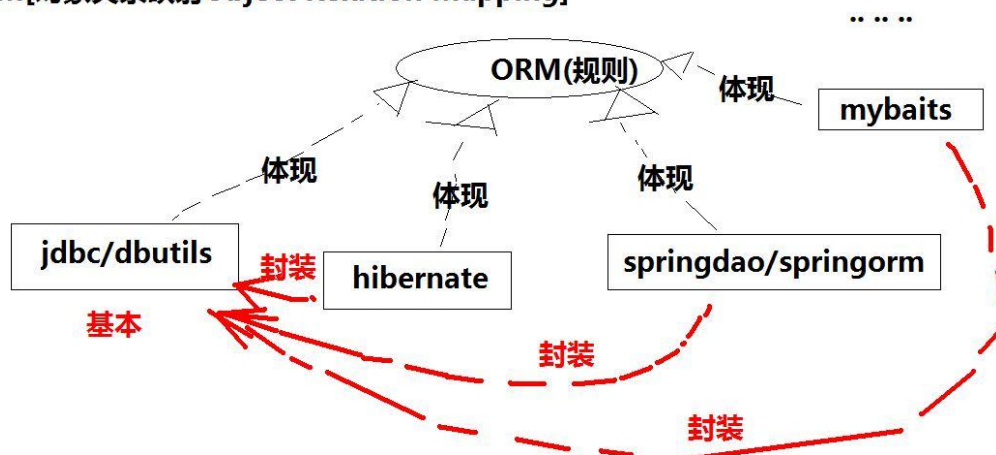
## 1. 什么是MyBatis

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code, 并且改名为MyBatis。是一个基于Java的持久层框架

## 2. 为什么我们要用Mybatis?

无论是Mybatis、Hibernate都是ORM的一种实现框架，都是对JDBC的一种封装！

## ORM[对象关系映射Object Relation Mapping]



[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

到目前为止，我们已经在持久层中学了几种技术了...

- Hibernate
- jdbc
- SpringDAO

那我们为啥还要学Mybatis呢??? 现在Mybatis在业内大行其道，那为啥他能那么火呢??

Hibernate是一个比较老旧的框架，用过他的同学都知道，只要你会用，用起来十分舒服...啥sql代码都不用写...但是呢，它也是有的缺点：：处理复杂业务时，灵活度差, 复杂的HQL难写难理解，例如多表查询的HQL语句

而JDBC很容易理解，就那么几个固定的步骤，就是开发起来太麻烦了，因为什么都要我们自己干..

我们可以认为，**Mybatis**就是**jdbc**和**Hibernate**之间的一个平衡点...毕竟现在业界都是用这个框架，我们也不能不学呀！

## 3. Mybatis快速入门

其实我们已经学过了Hibernate了，对于Mybatis入门其实就非常类似的。因此就很简单就能掌握基本的开发了...

### 3.1 导入开发包

更新：如果用Maven的同学，这里引入maven依赖就好了

导入Mybatis开发包

- mybatis-3.1.1.jar
- commons-logging-1.1.1.jar
- log4j-1.2.16.jar
- cglib-2.2.2.jar

- asm-3.3.1.jar

导入mysql/oracle开发包

- mysql-connector-java-5.1.7-bin.jar
- Oracle 11g 11.2.0.1.0 JDBC\_ojdbc6.jar

## 3.2 准备测试工作

创建一张表

```
create table students(  
    id int(5) primary key,  
    name varchar(10),  
    sal double(8,2)  
);
```

创建实体:

```
/**  
 * Created by ozc on 2017/7/21.  
 */  
  
public class Student {  
    private Integer id;  
    private String name;  
    private Double sal;  
  
    public Student() {  
    }  
  
    public Integer getId() {  
        return id;  
    }  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public Double getSal() {  
        return sal;  
    }  
}
```

```

    }

    public void setSal(Double sal) {
        this.sal = sal;
    }
}

```

### 3.3 创建mybatis配置文件

创建mybatis的配置文件，配置数据库的信息....数据库我们可以配置多个，但是默认的只能用一个...

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <!-- 加载类路径下的属性文件 -->
    <properties resource="db.properties"/>

    <!-- 设置一个默认的连接环境信息 -->
    <environments default="mysql_developer">
        <!-- 连接环境信息，取一个任意唯一的名字 -->
        <environment id="mysql_developer">
            <!-- mybatis使用jdbc事务管理方式 -->
            <transactionManager type="jdbc"/>
            <!-- mybatis使用连接池方式来获取连接 -->
            <dataSource type="pooled">
                <!-- 配置与数据库交互的4个必要属性 -->
                <property name="driver" value="${mysql.driver}"/>
                <property name="url" value="${mysql.url}"/>
                <property name="username" value="${mysql.username}"/>
                <property name="password" value="${mysql.password}"/>
            </dataSource>
        </environment>

        <!-- 连接环境信息，取一个任意唯一的名字 -->
        <environment id="oracle_developer">
            <!-- mybatis使用jdbc事务管理方式 -->
            <transactionManager type="jdbc"/>
            <!-- mybatis使用连接池方式来获取连接 -->
            <dataSource type="pooled">
                <!-- 配置与数据库交互的4个必要属性 -->
                <property name="driver" value="${oracle.driver}"/>
                <property name="url" value="${oracle.url}"/>
                <property name="username" value="${oracle.username}"/>
            </dataSource>
        </environment>
    </environments>

```

```

        <property name="password" value="${oracle.password}"/>
    </dataSource>
</environment>
</environments>

</configuration>

```

### 3.4 编写工具类测试是否获取到连接

使用Mybatis的API来创建一个工具类，通过mybatis配置文件与数据库的信息，得到Connection对象

```

package cn.itcast.javaee.mybatis.util;

import java.io.IOException;
import java.io.Reader;
import java.sql.Connection;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

/**
 * 工具类
 * @author AdminTC
 */
public class MybatisUtil {
    private static ThreadLocal<SqlSession> threadLocal = new
ThreadLocal<SqlSession>();
    private static SqlSessionFactory sqlSessionFactory;
    /**
     * 加载位于src/mybatis.xml配置文件
     */
    static{
        try {
            Reader reader = Resources.getResourceAsReader("mybatis.xml");
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
    /**
     * 禁止外界通过new方法创建
     */
    private MybatisUtil(){}
    /**
     * 获取SqlSession
     */

```



```

public static SqlSession getSqlSession(){
    //从当前线程中获取SqlSession对象
    SqlSession sqlSession = threadLocal.get();
    //如果SqlSession对象为空
    if(sqlSession == null){
        //在SqlSessionFactory非空的情况下，获取SqlSession对象
        sqlSession = sqlSessionFactory.openSession();
        //将SqlSession对象与当前线程绑定在一起
        threadLocal.set(sqlSession);
    }
    //返回SqlSession对象
    return sqlSession;
}
/**
 * 关闭SqlSession与当前线程分开
 */
public static void closeSqlSession(){
    //从当前线程中获取SqlSession对象
    SqlSession sqlSession = threadLocal.get();
    //如果SqlSession对象非空
    if(sqlSession != null){
        //关闭SqlSession对象
        sqlSession.close();
        //分开当前线程与SqlSession对象的关系，目的是让GC尽早回收
        threadLocal.remove();
    }
}
/**
 * 测试
 */
public static void main(String[] args) {
    Connection conn = MybatisUtil.getSqlSession().getConnection();
    System.out.println(conn!=null?"连接成功":"连接失败");
}
}

```

## 3.5 创建实体与映射关系文件

配置实体与表的映射关系

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!-- namespace属性是名称空间，必须唯一 -->
<mapper namespace="cn.javaee.mybatis.Student">

    <!-- resultMap标签:映射实体与表

```

```

    type属性：表示实体全路径名
    id属性：为实体与表的映射取一个任意的唯一的名字
-->
<resultMap type="student" id="studentMap">
    <!-- id标签：映射主键属性
        result标签：映射非主键属性
        property属性：实体的属性名
        column属性：表的字段名
    -->
    <id property="id" column="id"/>
    <result property="name" column="name"/>
    <result property="sal" column="sal"/>
</resultMap>

</mapper>

```

现在我们已经有了Mybatis的配置文件和表与实体之前的映射文件了，因此我们要将配置文件和映射文件关联起来

```

<mappers>
    <mapper resource="StudentMapper.xml"/>
</mappers>

```

在测试类上，我们是可获取到连接的

The screenshot shows an IDE with a project structure on the left and a code editor on the right. The code editor displays a Java class with a main method for testing database connectivity. Below the code editor, the 'Run' console shows the execution output, including log messages and a confirmation that the connection was successful.

```

64
65
66 * 测试
67 */
68 public static void main(String[] args) {
69     Connection conn = MybatisUtil.getSqlSession().getConnection();
70     System.out.println(conn!=null?"连接成功":"连接失败");
71 }
72
73
74

```

```

Run MybatisUtil
"X:\Program Files (x86)\Java\jdk1.7.0\bin\java" ...
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
连接成功
Process finished with exit code 0
http://blog.csdn.net/hon_3y

```

## 3.6 编写DAO

```

public class StudentDao {

    public void add(Student student) throws Exception {
        //得到连接对象
        SqlSession sqlSession = MybatisUtil.getSqlSession();
        sqlSession.insert();
    }
}

```

```

    }

    public static void main(String[] args) throws Exception {

        StudentDao studentDao = new StudentDao();

        Student student = new Student(1, "zhongfucheng", 10000D);
        studentDao.add(student);

    }
}

```

到现在为止，我们实体与表的映射文件仅仅映射了实体属性与表的字段的关系...

我们在Hibernate中如果想要插入数据什么的，只要调用save()方法就行了。Hibernate是自动化屏蔽掉了数据库的差异，而我们Mybatis是需要自己手动编写SQL代码的...

那么SQL代码是写在哪里的呢??? 明显地，我们作为一个框架，不可能在程序中写SQL，我们是在实体与表的映射文件中写的!

**Mybatis实体与表的映射文件中提供了insert标签【SQL代码片段】供我们使用**

```

//在JDBC中我们通常使用?号作为占位符，而在Mybatis中，我们是使用#{ }作为占位符
//parameterType我们指定了传入参数的类型
//#{ }实际上就是调用了Student属性的get方法

<insert id="add" parameterType="Student">

    INSERT INTO ZHONGFUCHENG.STUDENTS (ID, NAME, SAL) VALUES (#{id},#{name},#{
{sal}});

</insert>

```

在程序中调用映射文件的SQL代码片段

```

public void add(Student student) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID，就可以调用对应的映射文件中的SQL
        sqlSession.insert("StudentID.add", student);
        sqlSession.commit();
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

```

```
}  
}
```

值得注意的是：**Mybatis**中的事务是默认开启的，因此我们在完成操作以后，需要我们手动去提交事务！

## 4. Mybatis工作流程

---

- 通过Reader对象读取Mybatis映射文件
- 通过SqlSessionFactoryBuilder对象创建SqlSessionFactory对象
- 获取当前线程的SqlSession
- 事务默认开启
- 通过SqlSession读取映射文件中的操作编号，从而读取SQL语句
- 提交事务
- 关闭资源



加油

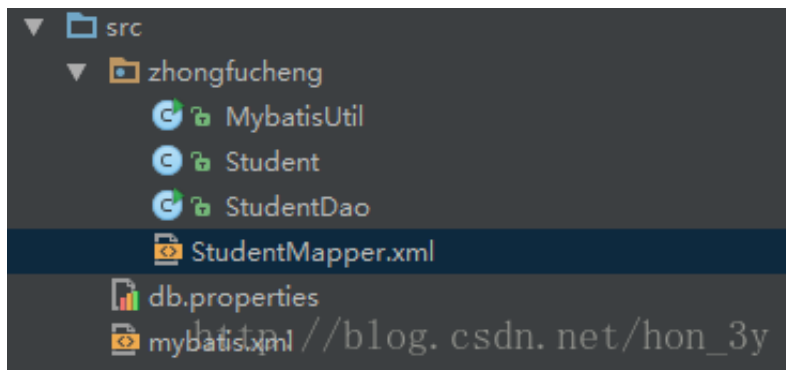


如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

## 5. 完成CRUD操作

我们在上面中已经简单知道了Mybatis是怎么使用的以及工作流程了，这次我们使用Mybatis来完成CRUD的操作，再次巩固Mybatis的开发步骤以及一些细节

包与类之间的结构



### 5.1 增加学生

配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
    <!-- 加载类路径下的属性文件 -->
    <properties resource="db.properties"/>

    <!-- 设置一个默认的连接环境信息 -->
    <environments default="mysql_developer">
        <!-- 连接环境信息，取一个任意唯一的名字 -->
        <environment id="mysql_developer">
            <!-- mybatis使用jdbc事务管理方式 -->
            <transactionManager type="jdbc"/>
            <!-- mybatis使用连接池方式来获取连接 -->
            <dataSource type="pooled">
                <!-- 配置与数据库交互的4个必要属性 -->
                <property name="driver" value="${mysql.driver}"/>
                <property name="url" value="${mysql.url}"/>
                <property name="username" value="${mysql.username}"/>
                <property name="password" value="${mysql.password}"/>
            </dataSource>
        </environment>

        <!-- 连接环境信息，取一个任意唯一的名字 -->
        <environment id="oracle_developer">
            <!-- mybatis使用jdbc事务管理方式 -->
```

```

<transactionManager type="jdbc"/>
<!-- mybatis使用连接池方式来获取连接 -->
<dataSource type="pooled">
    <!-- 配置与数据库交互的4个必要属性 -->
    <property name="driver" value="${oracle.driver}"/>
    <property name="url" value="${oracle.url}"/>
    <property name="username" value="${oracle.username}"/>
    <property name="password" value="${oracle.password}"/>
</dataSource>
</environment>
</environments>
<mappers>
    <mapper resource="zhongfucheng/StudentMapper.xml"/>
</mappers>
</configuration>

```

## 映射文件

```

<!-- namespace属性是名称空间，必须唯一 -->
<mapper namespace="StudentID">

    <!-- resultMap标签：映射实体与表
        type属性：表示实体全路径名
        id属性：为实体与表的映射取一个任意的唯一的名字
    -->
    <resultMap type="zhongfucheng.Student" id="studentMap">
        <!-- id标签：映射主键属性
            result标签：映射非主键属性
            property属性：实体的属性名
            column属性：表的字段名
        -->
        <id property="id" column="id"/>
        <result property="name" column="name"/>
        <result property="sal" column="sal"/>
    </resultMap>

    <insert id="add" parameterType="zhongfucheng.Student">
        INSERT INTO ZHONGFUCHENG.STUDENTS (ID, NAME, SAL) VALUES (#{id},#{name},#
{sal});
    </insert>

</mapper>

```

## 插入数据

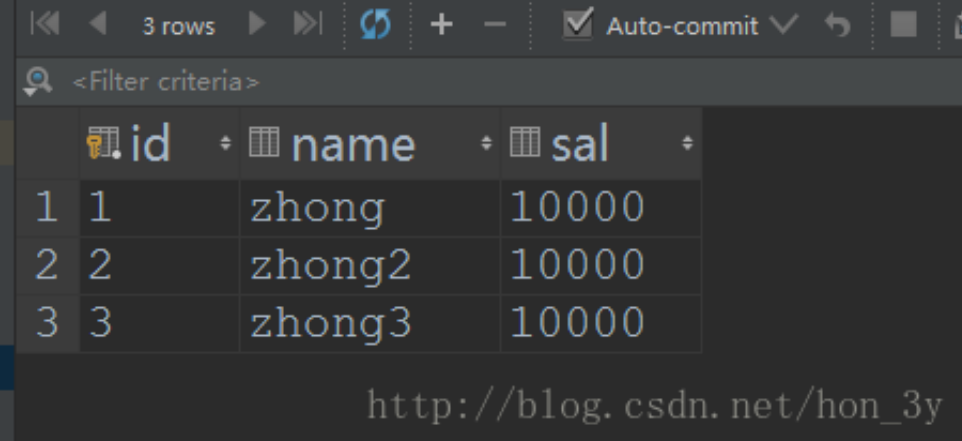
```

public class StudentDao {

    public void add(Student student) throws Exception {
        //得到连接对象
        SqlSession sqlSession = MybatisUtil.getSqlSession();
        try{
            //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
            sqlSession.insert("StudentID.add", student);
            sqlSession.commit();
        }catch(Exception e){
            e.printStackTrace();
            sqlSession.rollback();
            throw e;
        }finally{
            MybatisUtil.closeSqlSession();
        }
    }

    public static void main(String[] args) throws Exception {
        StudentDao studentDao = new StudentDao();
        Student student = new Student(3, "zhong3", 10000D);
        studentDao.add(student);
    }
}

```



	id	name	sal
1	1	zhong	10000
2	2	zhong2	10000
3	3	zhong3	10000

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 5.2根据ID查询数据

增加select标签

```

<!--
    查询根据id
    resultMap这个属性代表是返回值类型，返回值的类型是Student，就是上面实体类型
-->
<select id="findById" parameterType="int" resultMap="studentMap">
    SELECT * FROM STUDENTS WHERE id = #{id};
</select>

```

查询出来的结果是一个Student对象，我们调用SelectOne方法

```

public Student findById(int id) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID，就可以调用对应的映射文件中的SQL
        return sqlSession.selectOne("StudentID.findById", id);
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    Student student = studentDao.findById(1);
    System.out.println(student.getName());
}

```

```

StudentDao
"X:\Program Files (x86)\Java\jdk1.7.0\bin\java" ...
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
zhong

Process finished with exit code 0

```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 5.3 查询所有数据



```

<!--
    查询所有数据
    返回值类型讲道理是List<Student>的，但我们只要写集合中的类型就行了
-->
<select id="findAll" resultMap="studentMap">
    SELECT * FROM STUDENTS;
</select>

```

我们查询出来的结果不单单只有一个对象了，因此我们使用的是SelectList这个方法

```

public List<Student> findAll() throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID，就可以调用对应的映射文件中的SQL
        return sqlSession.selectList("StudentID.findAll");
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    List<Student> students = studentDao.findAll();
    System.out.println(students.size());
}

```

## 5.4 根据id删除

```

<!--根据id删除-->
<delete id="delete" parameterType="int">
    DELETE FROM STUDENTS WHERE id=#{id};
</delete>

```

调用delete方法删除

```

public void delete(int id ) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID，就可以调用对应的映射文件中的SQL
    }
}

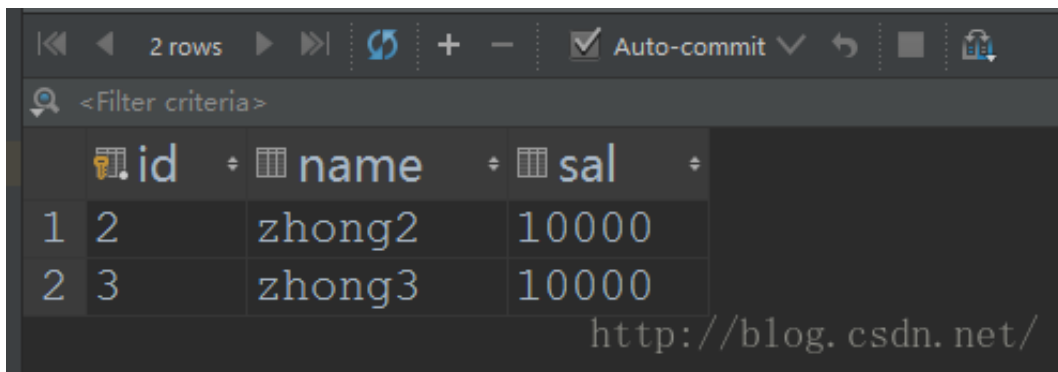
```

```

        sqlSession.delete("StudentID.delete", id);
        sqlSession.commit();
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    studentDao.delete(1);
}
}

```



id	name	sal
2	zhong2	10000
3	zhong3	10000

http://blog.csdn.net/

## 5.5 修改

```

<!--更新-->
<update id="update" parameterType="zhongfucheng.Student">

    update students set name=#{name},sal=#{sal} where id=#{id};

</update>

```

查询出对应的对象，对其进行修改

```

public void update(Student student ) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
        sqlSession.update("StudentID.update", student);
        sqlSession.commit();
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }
}

```

```

    }finally{
        MybatisUtil.closeSqlSession();
    }
}
public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    Student student = studentDao.findById(2);
    student.setName("fucheng");
    student.setSal(2000D);
    studentDao.update(student);
}

```

	id	name	sal
1	2	fucheng	2000
2	3	zhong3	10000

## 5.6 小细节

<!--

注意：这个insert/update/delete标签只是一个模板，在做操作时，其实是以SQL语句为核心的  
 即在做增/删/时，insert/update/delete标签可通用，  
 但做查询时只能用select标签  
 我们提倡什么操作就用什么标签

-->

## 5.7 Mybatis分页

分页是一个非常实用的技术点，我们也来学习一下使用Mybatis是怎么分页的...

我们的分页是需要多个参数的，并不是像我们之前的例子中只有一个参数。当需要接收多个参数的时候，我们使用Map集合来装载！

```

public List<Student> pagination(int start ,int end) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID，就可以调用对应的映射文件中的SQL
    }
}

```

```

    /**
     * 由于我们的参数超过了两个，而方法中只有一个Object参数收集
     * 因此我们使用Map集合来装载我们的参数
     */
    Map<String, Object> map = new HashMap();
    map.put("start", start);
    map.put("end", end);
    return sqlSession.selectList("StudentID.pagination", map);
} catch (Exception e) {
    e.printStackTrace();
    sqlSession.rollback();
    throw e;
} finally {
    MybatisUtil.closeSqlSession();
}
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    List<Student> students = studentDao.pagination(0, 3);
    for (Student student : students) {

        System.out.println(student.getId());

    }

}
}

```

那么在实体与表映射文件中，我们接收的参数就是map集合

```

<!--分页查询-->
<select id="pagination" parameterType="map" resultMap="studentMap">

    /*根据key自动找到对应Map集合的value*/
    select * from students limit #{start},#{end};

</select>

```

```

"X:\Program Files (x86)\Java\jdk1.7.0\bin\java" ...
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory)
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2
3
4
Process finished with exit code 0

```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 6. 动态SQL

何为动态SQL?? 回顾一下我们之前写的SSH项目中, 有多条件查询的情况, 如下图

编号	姓名	薪水	
<input type="text" value="不限"/>	<input type="text" value="不限"/>	<input type="text" value="不限"/>	<input type="button" value="搜索"/>
	哈哈	7000	

**select \* from students where name='哈哈'**

**select \* from students where sal=7000**

**select \* from students where name='哈哈' and sal=7000**

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

我们当时刚开始做的时候, 是需要在Controller中判断SQL是否已经有条件了, 因为SQL语句需要拼接起来....这样干的话, 就非常容易出错的。

如下的代码, 如果有多个条件的话, 那么拼接起来很容易出错!

```
public String listUI() {  
  
    //查询语句  
    String hql = "FROM Info i ";  
    List<Object> objectList = new ArrayList<>();  
  
    //根据info是否为null来判断是否是条件查询。如果info为空, 那么是查询所有。  
    if (info != null) {  
        if (StringUtils.isNotBlank(info.getTitle())) {  
            hql += "where i.title like ?";  
            objectList.add("%" + info.getTitle() + "%");  
        }  
    }  
    infoList = infoServiceImpl.findObjects(hql, objectList);  
    ActionContext.getContext().getContextMap().put("infoTypeMap",  
    Info.INFO_TYPE_MAP);  
    return "listUI";  
}
```

后来, 我们觉得这样不好, 于是就专门写了一个查询助手类:

```
package zhongfucheng.core.utils;  
  
import java.util.ArrayList;  
import java.util.List;
```

```

/**
 * Created by ozc on 2017/6/7.
 */
public class QueryHelper {

    private String fromClause = "";
    private String whereClause = "";
    private String orderByClause = "";
    private List<Object> objectList;

    public static String ORDER_BY_ASC = "asc";
    public static String ORDER_BY_DESC = "desc";

    //FROM子句只出现一次
    /**
     * 构建FROM字句，并设置查询哪张表
     * @param aClass 用户想要操作的类型
     * @param alias 别名
     */
    public QueryHelper(Class aClass, String alias) {
        fromClause = " FROM " + aClass.getSimpleName() + " " + alias;
    }
    //WHERE字句可以添加多个条件，但WHERE关键字只出现一次
    /**
     * 构建WHERE字句
     * @param condition
     * @param objects
     * @return
     */
    public QueryHelper addCondition(String condition, Object... objects) {
        //如果已经有字符了，那么就说明已经有WHERE关键字了
        if (whereClause.length() > 0) {
            whereClause += " AND " + condition;
        } else {
            whereClause += " WHERE" + condition;
        }
        //在添加查询条件的时候，?对应的查询条件值
        if (objects == null) {
            objectList = new ArrayList<>();
        }

        for (Object object : objects) {
            objectList.add(object);
        }

        return this;
    }

```

```

    }

    /**
     *
     * @param property 要排序的属性
     * @param order 是升序还是降序
     * @return
     */
    public QueryHelper orderBy(String property, String order) {

        //如果已经有字符了, 那么就说明已经有ORDER关键字了
        if (orderByClause.length() > 0) {
            orderByClause += " , " + property + " " + order;
        } else {
            orderByClause += " ORDER BY " + property + " " + order;
        }
        return this;
    }

    /**
     * 返回HQL语句
     */
    public String returnHQL() {
        return fromClause + whereClause + orderByClause;
    }

    /**
     * 得到参数列表
     * @return
     */
    public List<Object> getObjectList() {
        return objectList;
    }
}

```

这样一来的话, 我们就不用自己手动拼接了, 给我们的查询助手类去拼接就好了。

而如果我们使用Mybatis的话, 就可以免去查询助手类了。因为**Mybatis内部就有动态SQL的功能【动态SQL就是自动拼接SQL语句】**！

## 6.1 动态查询

```

<!--多条件查询【动态SQL】-->
<!--会自动组合成一个正常的WHERE字句-->
<!--name值会从map中寻找-->

<select id="findByCondition" resultMap="studentMap" parameterType="map">

    select * from students

```

```

<where>
  <if test="name!=null">
    and name=#{name}
  </if>
  <if test="sal!=null">
    and sal < #{sal}
  </if>
</where>

</select>

```

查询出来小于9000块的人

```

public List<Student> findByCondition(String name, Double sal) throws
Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
        /**
         * 由于我们的参数超过了两个, 而方法中只有一个Object参数收集
         * 因此我们可以使用Map集合来装载我们的参数
         */
        Map<String, Object> map = new HashMap();
        map.put("name", name);
        map.put("sal", sal);
        return sqlSession.selectList("StudentID.findByCondition", map);
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    List<Student> students = studentDao.findByCondition(null, 9000D);
    for (Student student : students) {
        System.out.println(student.getId() + "---" + student.getName() +
"----" + student.getSal());
    }
}

```



```
studentDao
"X:\Program Files (x86)\Java\jdk1.7.0\bin\java" ...
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2---fucheng---2000.0
4---zhong2---4000.0
5---zhong1---5000.0
6---zhong4---6000.0
7---zhong---7000.0
8---zhong---3000.0
9---zhong---3000.0

Process finished with exit code 0
```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 6.2 动态更新

更新条件不确定，需要根据情况产生SQL语法

//name为非null的情况

update students set name=? where id=?

//sal为非null的情况

update students set sal=? where id=?

//name和sal都非null的情况

update students set name=?,sal=? where id=?

```
update students
<set>
    <if test="name!=null">
        name=#{name},
    </if>
    <if test="sal!=null">
        sal=#{sal},
    </if>
</set>
where id=#{id}
```

update students set name=? and sal=? where id=?

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

```
<!--动态更新-->
<!--不要忘了逗号-->
<update id="updateByConditions" parameterType="map">

    update students
    <set>
        <if test="name!=null">
            name = #{name},
        </if>
        <if test="sal!=null">
            sal = #{sal},
        </if>
    </set>
    where id = #{id}
</update>
```

给出三个更新的字段

```

    public void updateByConditions(int id,String name,Double sal) throws
Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
        /**
         * 由于我们的参数超过了两个, 而方法中只有一个Object参数收集
         * 因此我们使用Map集合来装载我们的参数
         */
        Map<String, Object> map = new HashMap();
        map.put("id", id);
        map.put("name", name);
        map.put("sal", sal);
        sqlSession.update("StudentID.updateByConditions", map);
        sqlSession.commit();
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    studentDao.updateByConditions(2,"haha",500D);
}

```

8 rows

<Filter criteria>

	id	name	sal
1	2	haha	500
2	3	zhong3	10000
3	4	zhong2	4000
4	5	zhong1	5000
5	6	zhong4	6000
6	7	zhong	7000
7	8	zhong	3000
8	9	zhong	3000

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 6.3 动态删除

根据ID查询学生

delete from students where id in ( 1, 3, 5, 7 )

```
delete from students where id in
<!--
  <foreach collection="array" open="(" close=")" separator=","
    item="ids">
    ${ids}
  </foreach>
-->
  <foreach
    collection="list"
    open="(" close=")"
    separator=","
    item="ids">
    #{ids}
  </foreach>
```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

以前我们使用JDBC也好，Hibernate也好，想要批量删除的时候，总是使用的是循环删除。而我们现在使用的是Mybatis，SQL语句是自己写的。所以我们可以写下如下的SQL来进行删除

```
delete from students where id in (?,?,?,?);
```

而我们的Mybatis又支持动态SQL,所以删除起来就非常方便了！

```
<delete id="deleteByConditions" parameterType="int">
```

```
<!-- foreach用于迭代数组元素
    open表示开始符号
```

close表示结束符合  
 separator表示元素间的分隔符  
 item表示迭代的数组，属性值可以任意，但提倡与方法的数组名相同  
 #{ids}表示数组中的每个元素值

```
-->
delete from students where id in
<foreach collection="array" open="(" close=")" separator="," item="ids">
  #{ids}
</foreach>

</delete>
```

删除编号为2, 3, 4的记录

```
public void deleteByConditions(int... ids) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
        /**
         * 由于我们的参数超过了两个, 而方法中只有一个Object参数收集
         * 因此我们使用Map集合来装载我们的参数
         */
        sqlSession.delete("StudentID.deleteByConditions", ids);
        sqlSession.commit();
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    studentDao.deleteByConditions(2,3,4);
}
```

StudentMapper.xml x StudentDao.java x zhongfucheng.students x			
5 rows			
<Filter criteria>			
	id	name	sal
1	5	zhong1	5000
2	6	zhong4	6000
3	7	zhong	7000
4	8	zhong	3000
5	9	zhong	3000

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 6.4 动态插入

我们要想动态插入的话，就比其他的DML语句稍微复杂一点，因为它有两部分是不确定的，平常的SQL语句是这样的：

```
insert into student(id,name,sal) values(?,?,?)
```

//id,name,sal!=null  
insert into students(id,name,sal) values(?,?,?)

//id,name!=null  
insert into students(id,name) values(?,?)

//id,sal!=null  
insert into students(id,sal) values(?,?)

```
<sql id="key">
  <trim suffixOverrides=",">
    <if test="id!=null">
      id,
    </if>
    <if test="name!=null">
      name,
    </if>
    <if test="sal!=null">
      sal,
    </if>
  </trim>
</sql>
<sql id="value">
  <trim suffixOverrides=",">
    <if test="id!=null">
      #{id},
    </if>
    <if test="name!=null">
      #{name},
    </if>
    <if test="sal!=null">
      #{sal},
    </if>
  </trim>
</sql>

<insert id="dynamicSQLwithInsert" parameterType="cn.itcast.javaee.mybatis.app14.Student">
  insert into students(<include refid="key"/>) values(<include refid="value"/>)
</insert>
```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

SQL代码块是不能像之前那样帮我们自动去除多余的逗号的，因此我们需要使用trim标签来自己手动去除...

编写insertSQL语句的时候，不要忘了写()括号。

```

<!--SQL片段默认是不帮我们自动生成合适的SQL, 因此需要我们自己手动除去逗号-->
<sql id="key">
    <trim suffixOverrides=", ">
        <if test="id!=null">
            id,
        </if>

        <if test="id!=null">
            name,
        </if>

        <if test="id!=null">
            sal,
        </if>
    </trim>
</sql>

<sql id="value">
    <trim suffixOverrides=", ">
        <if test="id!=null">
            #{id},
        </if>

        <if test="id!=null">
            #{name},
        </if>

        <if test="id!=null">
            #{sal},
        </if>
    </trim>
</sql>
<!--动态插入-->
<insert id="insertByConditions" parameterType="zhongfucheng.Student">

    insert into students (<include refid="key"/>) values
        (<include refid="value"/>)

</insert>

```

测试三个不同内容的数据

```

public void insertByConditions(Student student) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{
        //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
        sqlSession.insert("StudentID.insertByConditions", student);
    }
}

```

```

        sqlSession.commit();
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    studentDao.insertByConditions(new Student(55, null, null)); //name和sal
    studentDao.insertByConditions(new Student(66, "haxi", null)); //sal为空
    studentDao.insertByConditions(new Student(77, null, 3999d)); //name为空
}

```

为空

	id	name	sal
1	5	zhong1	5000
2	6	zhong4	6000
3	7	zhong	7000
4	8	zhong	3000
5	9	zhong	3000
6	22	<null>	<null>
7	33	haxi	<null>
8	55	<null>	<null>
9	66	haxi	<null>
10	77	<null>	3999

## 7. 入门总结

- Mybatis的准备工作与Hibernate差不多，都需要一个总配置文件、一个映射文件。
- Mybatis的SQLSession工具类使用ThreadLocal来对线程中的Session来进行管理。
- **Mybatis**的事务默认是开启的，需要我们手动去提交事务。
- **Mybatis**的SQL语句是需要手写的，在程序中通过映射文件的命名空间.sql语句的id来进行调用！
- 在Mybatis中，增删改查都是Mybatis需要我们自己写SQL语句的，然后在程序中调用即可了。SQL由于是我们自己写的，于是就相对Hibernate灵活一些。
- 如果需要传入多个参数的话，那么我们一般在映射文件中用Map来接收。

- 由于我们在开发中会经常用到条件查询，在之前，我们是使用查询助手来帮我们完成对SQL的拼接的。而Mybatis的话，我们是自己手写SQL代码的。
- **Mybatis**也支持一些判断标签，于是我们就可以通过这些标签来完成动态**CRUD**的操作了。
- 值得注意的是，我们的sql片段代码是需要我们自己手动去分割，号的。

加油~



## Mybatis配置信息

---

### 1. 映射文件

---

在mapper.xml文件中配置很多的sql语句，执行每个sql语句时，封装为MappedStatement对象，mapper.xml以statement为单位管理sql语句

Statement的实际位置就等于namespace+StatementId

#### 1.1 占位符



在Mybatis中，有两种占位符

- `#{}`  解析传递进来的参数数据
- `${}`  对传递进来的参数原样拼接在SQL中

## 1.2 主键生成策略

如果我们在Hibernate中，当我们插入数据的时候，我们是可以选择是UUID策略的...

那么在Mybatis是怎么做的呢？

### 1.2.1 UUID

```
<!-- mysql的uuid生成主键 -->
<insert id="insertUser" parameterType="cn.mybatis.po.User">
    <selectKey keyProperty="id" order="BEFORE" resultType="string">
        select uuid()
    </selectKey>

    INSERT INTO USER(id,username,birthday,sex,address) VALUES(#{id},#
{username},#{birthday},#{sex},#{address})
</insert>
```

## 1.3 主键返回

如果我们一般插入数据的话，如果我们想要知道刚刚插入的数据的主键是多少，我们可以通过以下的方式来获取

需求：user对象插入到数据库后，新记录的主键要通过user对象返回，通过user获取主键值。

解决思路：通过LAST\_INSERT\_ID()获取刚插入记录的自增主键值，在insert语句执行后，执行select LAST\_INSERT\_ID()就可以获取自增主键。

mysql:

```
<insert id="insertUser" parameterType="cn.mybatis.po.User">
    <selectKey keyProperty="id" order="AFTER" resultType="int">
        select LAST_INSERT_ID()
    </selectKey>

    INSERT INTO USER(username,birthday,sex,address) VALUES(#{username},#
{birthday},#{sex},#{address})
</insert>
```

oracle: 先查询序列得到主键，将主键设置到user对象中，将user对象插入数据库。

```

<!-- oracle
在执行insert之前执行select 序列.nextval() from dual取出序列最大值，将值设置到user对象
的id属性
-->
<insert id="insertUser" parameterType="cn.mybatis.po.User">
  <selectKey keyProperty="id" order="BEFORE" resultType="int">
    select 序列.nextval() from dual
  </selectKey>

  INSERT INTO USER(id,username,birthday,sex,address) VALUES( 序列.nextval(),#{
username},#{birthday},#{sex},#{address})
</insert>

```

也可以在 `select` 标签下写以下的属性：

```

< select useGeneratedKeys="true" keyProperty="id" keyColumn="id" />

```

## 1.4 resultMap

有的时候，我们看别的映射文件，可能看不到以下这么一段代码：

```

<resultMap id="userListResultMap" type="user" >
  <!-- 列名
  id_,username_,birthday_
  id: 要映射结果集的唯一标识，称为主键
  column: 结果集的列名
  property: type指定的哪个属性中
  -->
  <id column="id_" property="id"/>
  <!-- result就是普通列的映射配置 -->
  <result column="username_" property="username"/>
  <result column="birthday_" property="birthday"/>

</resultMap>

```

因为，如果我们的数据表的字段和JavaBean的属性名称是相同时，我们就不用上面那段代码了。

**Mybatis会自动帮我们返回的结果进行封装成JavaBean**

那当我们数据表的字段和JavaBean的属性名称不是相同时，我们就需要使用resultMap，也就是上面那段代码

- 当然了，在正常情况下列名和JavaBean的属性名一般都是不同的，因此还是需要resultMap的。

## 1.5 resultMap和resultType区别

resultType：指定输出结果的类型（pojo、简单类型、hashmap..），将sql查询结果映射为java对象

。

- 使用resultType注意：**sql查询的列名要和resultType指定pojo的属性名相同**，指定相同 属性方可**映射成功**，如果sql查询的列名要和resultType指定pojo的属性名全部不相同，list中无法创建pojo对象的。

resultMap：将sql查询结果映射为java对象。

- 如果sql查询列名和最终要映射的pojo的属性名不一致，使用**resultMap**将列名和pojo的属性名做一个对应关系（列名和属性名映射配置）

## 1.6 使用resultMap

```
<resultMap id="userListResultMap" type="user" >
  <!-- 列名
  id_,username_,birthday_
  id: 要映射结果集的唯一标识，称为主键
  column: 结果集的列名
  property: type指定的哪个属性中
  -->
  <id column="id_" property="id"/>
  <!-- resultMap就是普通列的映射配置 -->
  <result column="username_" property="username"/>
  <result column="birthday_" property="birthday"/>
</resultMap>
```

<!-- 使用resultMap作结果映射

resultMap: 如果引用resultMap的位置和resultMap的定义在同一个mapper.xml，直接使用resultMap的id，如果不在同一个mapper.xml要在resultMap的id前边加namespace

-->

```
<select id="findUserListResultMap" parameterType="userQueryVo" resultMap="userListResultMap">
```

```
  select id id_,username username_,birthday birthday_ from user where username like '%${userCustom.username}'
```

```
</select>
```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

//查询用户，使用resultMap进行映射

```
public List<User> findUserListResultMap(UserQueryVo userQueryVo) throws Exception;
```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 1.7 resultMap和resultType用法总结

resultType：

- 作用：将查询结果按照**sql列名pojo属性名一致性**映射到pojo中。
- 场合：常见一些明细记录的展示，将关联查询信息全部展示在页面时，此时可直接使用resultType将每一条记录映射到pojo中，在前端页面遍历list（list中是pojo）即可。

resultMap：

- 使用**association**和**collection**完成一对一和一对多高级映射。

association:

- 作用：将关联查询信息映射到一个pojo类中。
- 场合：为了方便获取关联信息可以使用association将关联订单映射为pojo，比如：查询订单及关联用户信息。

collection:

- 作用：将关联查询信息映射到一个list集合中。
- 场合：为了方便获取关联信息可以使用collection将关联信息映射到list集合中，比如：查询用户权限范围模块和功能，可使用collection将模块和功能列表映射到list中。

Collection在前面好像并没有用过，下面就看一下它的用法：

Order与OrderDetails关系

```
package cn.mybatis.po;

import java.io.Serializable;
import java.util.Date;
import java.util.List;

public class Orders implements Serializable {
    private Integer id;

    private Integer userId;

    private String number;

    private Date createtime;

    private String note;

    //关联用户信息
    private User user;

    //订单明细
    private List<Orderdetail> orderdetails;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getUserId() {
        return userId;
    }
```

```

    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number == null ? null : number.trim();
    }

    public Date getCreatetime() {
        return createtime;
    }

    public void setCreatetime(Date createtime) {
        this.createtime = createtime;
    }

    public String getNote() {
        return note;
    }

    public void setNote(String note) {
        this.note = note == null ? null : note.trim();
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public List<Orderdetail> getOrderdetails() {
        return orderdetails;
    }

    public void setOrderdetails(List<Orderdetail> orderdetails) {
        this.orderdetails = orderdetails;
    }
}

```

## SQL语句

```
<!-- 一对多查询使用resultMap完成
查询订单关联查询订单明细
-->
<select id="findOrderAndOrderDetails" resultMap="orderAndOrderDetails" >
    SELECT
    orders.*,
    user.username,
    user.sex ,
    orderdetail.id orderdetail_id,
    orderdetail.items_num,
    orderdetail.items_id
FROM
    orders,
    USER,
    orderdetail
WHERE orders.user_id = user.id  AND orders.id = orderdetail.orders_id
</select>
```

## resultMap

```
<!-- 一对多，查询订单及订单明细 -->
<resultMap type="orders" id="orderAndOrderDetails"
extends="ordersUserResultMap">
    <!-- 映射订单信息，和用户信息，这里使用继承ordersUserResultMap -->

    <!-- 映射订单明细信息
    property: 要将关联信息映射到orders的哪个属性中
    ofType: 集合中pojo的类型
    -->
    <collection property="orderdetails" ofType="cn.mybatis.po.Orderdetail">
        <!-- id: 关联信息订单明细的唯一标识
        property: Orderdetail的属性名
        -->
        <id column="orderdetail_id" property="id"/>
        <result column="items_num" property="itemsNum"/>
        <result column="items_id" property="itemsId"/>
    </collection>
</resultMap>
```

一般地使用resultMap会多一点。

## 1.8Mybatis映射文件处理特殊字符

第一种方法：

- 用了转义字符把>和<替换掉就OK了

第二种方法：

- `<![CDATA[ ]]>`



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

## 2. 配置文件

### 2.1 别名

typeAliases别名：

别名	映射的类型
<u>_byte</u>	byte
<u>_long</u>	long
<u>_short</u>	short
<u>_int</u>	int
<u>_integer</u>	int
<u>_double</u>	double
<u>_float</u>	float
<u>_boolean</u>	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
<u>boolean</u>	Boolean
date	Date
decimal	<u>BigDecimal</u>
<u>bigdecimal</u>	<u>BigDecimal</u>

自定义别名：

```

<!-- 定义 别名 -->
<typeAliases>
  <!--
  单个别名的定义
  alias: 别名, type: 别名映射的类型  -->
  <!-- <typeAlias type="cn.mybatis.po.User" alias="user"/> -->
  <!-- 批量别名定义
  指定包路径, 自动扫描包下边的pojo, 定义别名, 别名默认为类名 (首字母小写或大写)
  -->
  <package name="cn.mybatis.po"/>

</typeAliases>

```

## 2.2Mapper加载

```

<mappers>

```



```

<!-- 通过resource引用mapper的映射文件 -->
<mapper resource="sqlmap/User.xml" />
<!-- <mapper resource="mapper/UserMapper.xml" /> -->
<!-- 通过class引用mapper接口
class: 配置mapper接口全限定名
要求: 需要mapper.xml和mapper.java同名并且在一个目录 中
-->
<!-- <mapper class="cn.mybatis.mapper.UserMapper"/> -->
<!-- 批量mapper配置
通过package进行自动扫描包下边的mapper接口,
要求: 需要mapper.xml和mapper.java同名并且在一个目录 中
-->
<package name="cn.mybatis.mapper"/>
</mappers>

```

## 2.3延迟加载

在进行数据查询时，为了提高数据库查询性能，尽量使用单表查询，因为单表查询比多表关联查询速度要快。

如果查询单表就可以满足需求，一开始先查询单表，当需要关联信息时，再关联查询，当需要关联信息再查询这个叫延迟加载。

在Mybatis中延迟加载就是在resultMap中配置具体的延迟加载..

设置项	描述	允许值	默认值
<u>lazyLoadingEnabled</u>	全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。	true   false	false
<u>aggressiveLazyLoading</u>	当设置为‘true’的时候,懒加载的对象可能被任何懒属性全部加载。否则，每个属性都 <b>按需加载</b> 。	true   false	true

在Mybatis的文件中配置全局延迟加载

```

<!-- 全局配置参数 -->
<settings>
  <!-- 延迟加载总开关 -->
  <setting name="lazyLoadingEnabled" value="true" />
  <!-- 设置按需加载 -->
  <setting name="aggressiveLazyLoading" value="false" />
</settings>

```

## 2.4 延迟加载测试

当需要用户时调用 **Orders**类中的**getUser()**方法执行延迟加载，向数据库发出sql。

由于是对User进行延迟加载，那么我们只要查询Orders相关的信息即可了

```
<!-- 一对一查询延迟加载
开始只查询订单，对用户信息进行延迟加载
-->
<select id="findOrderUserListLazyLoading"
resultMap="orderCustomLazyLoading">
    SELECT
        orders.*
    FROM
        orders
</select>
```

使用resultMap来配置延迟加载

```
<!-- 一对一查询延迟加载 的配置 -->
<resultMap type="orders" id="orderCustomLazyLoading">
    <!-- 完成了订单信息的映射配置 -->
    <!-- id: 订单关联用户查询的唯一标识 -->
    <id column="id" property="id" />
    <result column="user_id" property="userId" />
    <result column="number" property="number" />
    <result column="createtime" property="createtime" />
    <result column="note" property="note" />
    <!--

    配置用户信息的延迟加载
        select: 延迟加载执行的sql所在的statement的id, 如果不在同一个namespace需要加namespace
        sql: 根据用户id查询用户信息 【column就是参数】
        column: 关联查询的列
        property: 将关联查询的用户信息设置到Orders的哪个属性 -->

    <!--当需要user数据的时候，它就会把column所指定的user_id传递过去给
    cn.mybatis.mapper.UserMapper.findUserId作为参数来查询数据-->
    <association property="user"
        select="cn.mybatis.mapper.UserMapper.findUserId" column="user_id">
</association>

</resultMap>
```

```
// 一对一查询延迟加载
@Test
public void testFindOrderUserListLazyLoading() throws Exception {

    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 创建mapper代理对象
    OrdersMapperCustom ordersMapperCustom = sqlSession
        .getMapper(OrdersMapperCustom.class);

    // 调用方法
    List<Orders> list = ordersMapperCustom.findOrderUserListLazyLoading()

    //这里执行延迟加载，要发出sql
    User user = list.get(0).getUser();
    System.out.println(user);

}
http://blog.csdn.net/hon\_3y
```

### 3. 配置相关总结

- 在程序中调用的SQL语句是由映射文件的命令空间+sql片段的id所组成的。它内部会生成一个Statement对象的。
- 在使用别名的时候，可以指定包名，在使用总配置文件加载映射文件时，也可以指定包名。
- 主键如果需要返回的话，使用selectKey 标签即可。UUID也可以返回。在Oracle的话，是使用序列来返回自动增长的主键的。
- 占位符有两种，一种是解析传递进来的参数数据、一种是原样输出传递进来的数据。





如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜Java3y公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

## 关联映射

### 1. Mybatis 【多表连接】

我们在学习Hibernate的时候，如果表涉及到两张的话，那么我们是在映射文件中使用 `<set>..<many-to-one>` 等标签将其的映射属性关联起来的...那么在我们Mybatis中又怎么做呢？？

先来回顾一下我们SQL99的语法：

一) 内连接（等值连接）：查询客户姓名，订单编号，订单价格

```
-----  
select c.name,o.isbn,o.price  
from customers c inner join orders o  
where c.id = o.customers_id;  
-----
```

```
select c.name,o.isbn,o.price  
from customers c join orders o  
where c.id = o.customers_id;  
-----
```

```
select c.name,o.isbn,o.price  
from customers c,orders o  
where c.id = o.customers_id;  
-----
```

```
select c.name,o.isbn,o.price  
from customers c join orders o  
on c.id = o.customers_id;  
-----
```

注意：内连接（等值连接）只能查询出多张表中，连接字段相同的记录

## 二) 外连接：按客户分组，查询每个客户的姓名和订单数

左外连接：

```
select c.name,count(o.isbn)
from customers c left outer join orders o
on c.id = o.customers_id
group by c.name;
```

右外连接：

```
select c.name,count(o.isbn)
from orders o right outer join customers c
on c.id = o.customers_id
group by c.name;
```

注意：外连接既能查询出多张表中，连接字段相同的记录；又能根据一方，将另一方不符合相同记录强行查询出来

## 三) 自连接：求出AA的老板是EE

内自连接：

```
select users.ename,boss.ename
from emps users inner join emps boss
on users.mgr = boss.empno;
```

外自连接：

```
select users.ename,boss.ename
from emps users left outer join emps boss
on users.mgr = boss.empno;
```

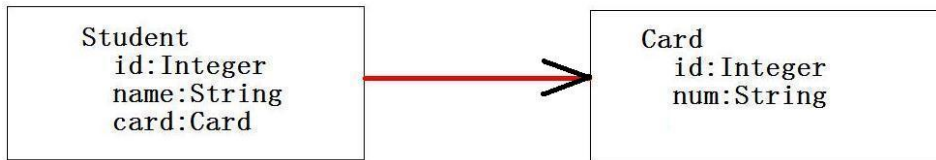
注意：自连接是将一张表，通过别名的方式，看作多张表后，再进行连接。  
这时的连接即可以采用内连接，又可以采用外连接

由于我们Mybatis中并没有像Hibernate这样全自动化的，因此我们是没有 `<set>` .. `<many-to-one>` 等标签的，我们还是使用手写SQL语句来使我们的关联属性连接起来...

## 1.1一对一

需求：学生和身份证

需求：查询1号学生及其身份证信息



```
mysql> select * from students;
+----+-----+-----+
| id | name | cid |
+----+-----+-----+
| 1  | 哈哈 | 1   |
+----+-----+-----+
1 row in set (0.30 sec)
```

```
mysql> select * from cards;
+----+-----+
| id | num |
+----+-----+
| 1  | 111 |
+----+-----+
1 row in set (0.00 sec)
```

作业：求出111身份证号对应的学生姓名？

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

### 1.1.1设计表：

```
--mysql

create table cards(
    cid int(5) primary key,
    cnum varchar(10)
);

create table students(
    sid int(5) primary key,
    sname varchar(10),
    scid int(5),
    constraint scid_fk foreign key(scid) references cards(cid)
);

insert into cards(cid,cnum) values(1,'111');
insert into students(sid,sname,scid) values(1,'哈哈',1);

select * from cards;
select * from students;
```

### 1.1.2实体

```
/**
 * 身份证(单方)
 * @author AdminTC
```

```

*/
public class Card {
    private Integer id;
    private String num;
    public Card(){}
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getNum() {
        return num;
    }
    public void setNum(String num) {
        this.num = num;
    }
}

```

```

/**
 * 学生(单方)
 * @author AdminTC
 */
public class Student {
    private Integer id;
    private String name;
    private Card card;//关联属性
    public Student(){}
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Card getCard() {
        return card;
    }
    public void setCard(Card card) {
        this.card = card;
    }
}

```

### 1.1.3 映射文件

由于我们有两个实体，因此我们会有两个映射文件

Student映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="studentNamespace">

    <resultMap type="zhongfucheng2.Student" id="studentMap">
        <id property="id" column="sid"/>
        <result property="name" column="sname"/>
    </resultMap>
</mapper>
```

Card映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="cardNamespace">

    <resultMap type="zhongfucheng2.Card" id="cardMap">
        <id property="id" column="cid"/>
        <result property="num" column="cnum"/>
    </resultMap>

</mapper>
```

### 1.1.4 DAO层

现在我想根据学生的编号查询学生的信息和身份证信息！

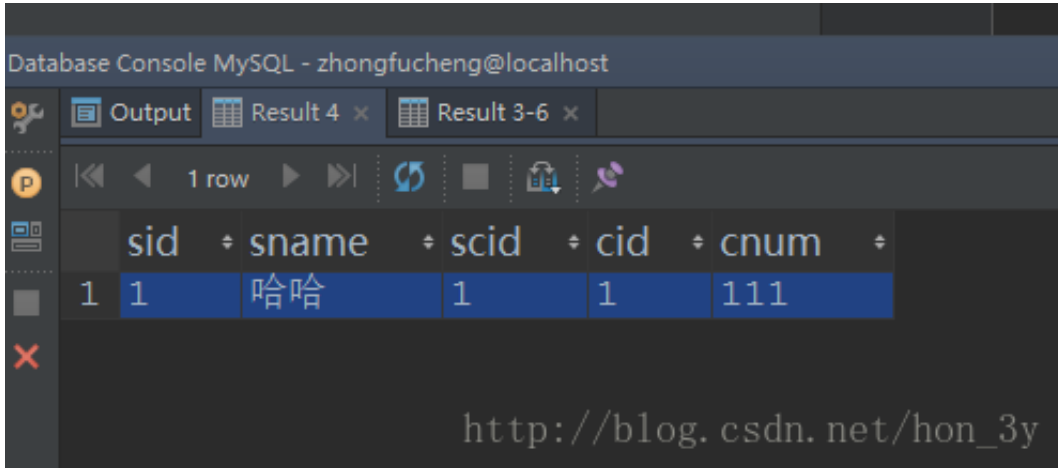
由于该查询着重是查询学生的信息，于是我们在学生的映射文件中写SQL语句

按照需求，我们写出来的SQL语句是这样子的。



```
select * from zhongfucheng.students s, zhongfucheng.cards c where c.cid = s.scid and sid=1;
```

我来看一下查询结果：



	sid	sname	scid	cid	cnum
1	1	哈哈	1	1	111

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

我们的实体与映射表中，**Student**实体是没有关联其他的字段的，仅仅是写出了该实体的自带的属性。

```
<resultMap type="zhongfucheng2.Student" id="studentMap">
  <id property="id" column="sid"/>
  <result property="name" column="sname"/>
</resultMap>
```

明显地，我们Student是不能封装返回的结果，因此我们需要将关联属性进行关联起来！

```
<resultMap type="zhongfucheng2.Student" id="studentMap">
  <id property="id" column="sid"/>
  <result property="name" column="sname"/>

  <!--
    property写的是在Student实体中写关联字段的属性变量名称
    resultMap写的是映射文件中的命名空间.id
  -->
  <association property="card" resultMap="cardNamespace.cardMap"/>
</resultMap>
```

我们关联了以后，Student实体就能够封装返回的结果了

```
<resultMap type="zhongfucheng2.Student" id="studentMap">
  <id property="id" column="sid"/>
  <result property="name" column="sname"/>

  <!--
    property写的是在Student实体中写关联字段的属性变量名称
```

```

    resultMap写的是映射文件中的命名空间.id
-->
    <association property="card" resultMap="cardNamespace.cardMap"/>
</resultMap>

<select id="findById" parameterType="int" resultMap="studentMap">
    select * from zhongfucheng.students s, zhongfucheng.cards c where c.cid =
s.scid and sid=#{id};
</select>

```

查询编号为1的学生信息【包括身份证编号】

```

public Student findById(int id) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{

        return sqlSession.selectOne("studentNamespace.findById", id);

        /* sqlSession.commit();*/
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    Student student = studentDao.findById(1);

    System.out.println(student.getId() + "----" + student.getName() + "----"
    + student.getCard().getNum());

}

```

```

StudentDao (1)
"C:\Program Files (x86)\Java\jdk1.7.0\bin\java" -Didea.launcher.port=7533 "-Didea.launcher.bin.path=X:\开发软件\IntelliJ
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
1---哈哈---111

Process finished with exit code 0

```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

## 1.2 一对多

需求：一个班级有多个学生,查询java学科有哪些学生信息

需求: 查询java班级有哪些【学生】  
查询哈哈属于哪个【班级】

对象世界



关系世界

```
mysql> select * from grades;
+----+-----+
| gid | gname |
+----+-----+
| 1   | java  |
+----+-----+
1 row in set (0.00 sec)
```

```
mysql> select * from students;
+----+-----+----+
| sid | sname | sgid |
+----+-----+----+
| 1   | 哈哈  | 1    |
| 2   | 呵呵  | 1    |
+----+-----+----+
```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 1.2.1设计数据库表

```
create table grades(
    gid int(5) primary key,
    gname varchar(10)
);

create table students(
    sid int(5) primary key,
    sname varchar(10),
    sgid int(5),
    constraint sgid_fk foreign key(sgid) references grades(gid)
);

insert into grades(gid,gname) values(1,'java');

insert into students(sid,sname,sgid) values(1,'哈哈',1);
insert into students(sid,sname,sgid) values(2,'呵呵',1);

select * from grades;
select * from students;
```

## 1.2.2实体

```
package zhongfucheng2;
```

```

import java.util.ArrayList;
import java.util.List;

/**
 * 学科(单方)
 * @author AdminTC
 */
public class Grade {
    private Integer id;
    private String name;
    private List<Student> studentList = new ArrayList<Student>();//关联属性
    public Grade(){
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<Student> getStudentList() {
        return studentList;
    }
    public void setStudentList(List<Student> studentList) {
        this.studentList = studentList;
    }
}

```

```

package zhongfucheng2;

/**
 * 学生(多方)
 * @author AdminTC
 */
public class Student {
    private Integer id;
    private String name;
    private Grade grade;//关联属性
    public Student(){
    }
    public Integer getId() {
        return id;
    }
}

```

```

public void setId(Integer id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public Grade getGrade() {
    return grade;
}
public void setGrade(Grade grade) {
    this.grade = grade;
}
}

```

### 1.2.3映射文件SQL语句

```

<mapper namespace="studentNamespace">

    <resultMap type="zhongfucheng2.Student" id="studentMap">
        <id property="id" column="sid"/>
        <result property="name" column="sname"/>
    </resultMap>

    <!--查询选修的java学科有多少位学生-->

    <!--由于我们只要查询学生的名字，而我们的实体studentMap可以封装学生的名字，那么我们返回
    studentMap即可，并不需要再关联到学科表-->
    <select id="findByGrade" parameterType="string" resultMap="studentMap">

        select s.sname,s.sid from zhongfucheng.students s, zhongfucheng.grades g
        WHERE s.sgid=g.gid and g.gname=#{name};

    </select>

</mapper>

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

```

```

<mapper namespace="gradeNamespace">

    <resultMap type="zhongfucheng2.Grade" id="gradeMap">
        <id property="id" column="gid"/>
        <result property="name" column="gname"/>
    </resultMap>
</mapper>

```

## 1.2.4DAO

```

public List<Student> findByGrade(String grade) throws Exception {
    //得到连接对象
    SqlSession sqlSession = MybatisUtil.getSqlSession();
    try{

        return sqlSession.selectList("studentNamespace.findByGrade",
grade);
        /* sqlSession.commit();*/
    }catch(Exception e){
        e.printStackTrace();
        sqlSession.rollback();
        throw e;
    }finally{
        MybatisUtil.closeSqlSession();
    }
}

public static void main(String[] args) throws Exception {
    StudentDao studentDao = new StudentDao();
    List<Student> student = studentDao.findByGrade("java");

    for (Student student1 : student) {
        System.out.println(student1.getName());
    }
}

```

```
studentDao()
"X:\Program Files (x86)\Java\jdk1.7.0\bin\java" ...
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory)
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
哈哈
呵呵

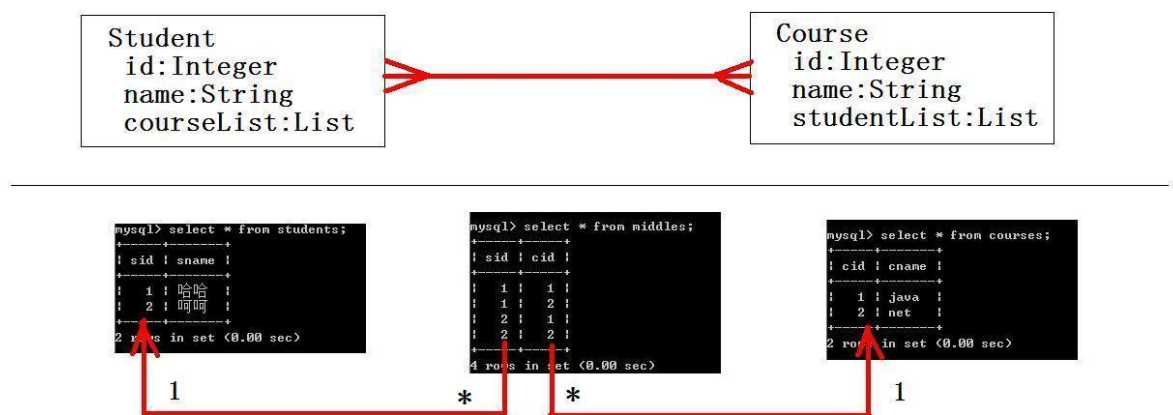
Process finished with exit code 0
|

http://blog.csdn.net/hon_3y
```

## 1.3多对多

需求：学生和课程

需求： 查询哈哈选学的【课程】  
查询java课程有哪些【学生】



http://blog.csdn.net/hon\_3y

### 1.3.1 数据库表

```
create table students(
    sid int(5) primary key,
    sname varchar(10)
);

create table courses(
    cid int(5) primary key,
    cname varchar(10)
);

create table middles(
    msid int(5),
    mcid int(5),
    primary key(msid,mcid)
);
```



```

insert into students(sid,sname) values(1,'哈哈');
insert into students(sid,sname) values(2,'呵呵');

insert into courses(cid,cname) values(1,'java');
insert into courses(cid,cname) values(2,'android');

insert into middles(msid,mcid) values(1,1);
insert into middles(msid,mcid) values(1,2);
insert into middles(msid,mcid) values(2,1);
insert into middles(msid,mcid) values(2,2);

select * from students;
select * from courses;
select * from middles;

```

### 1.3.2 实体

```

package cn.itcast.javaee.mybatis.many2many;

import java.util.ArrayList;
import java.util.List;

/**
 * 课程(多方)
 * @author AdminTC
 */
public class Course {
    private Integer id;
    private String name;
    private List<Student> studentList = new ArrayList<Student>(); //关联属性
    public Course(){}
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<Student> getStudentList() {
        return studentList;
    }
    public void setStudentList(List<Student> studentList) {

```

```

        this.studentList = studentList;
    }
}

```

```

package cn.itcast.javaee.mybatis.many2many;

import java.util.ArrayList;
import java.util.List;

/**
 * 学生(多方)
 * @author AdminTC
 */
public class Student {
    private Integer id;
    private String name;
    private List<Course> courseList = new ArrayList<Course>(); //关联属性
    public Student() {}
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<Course> getCourseList() {
        return courseList;
    }
    public void setCourseList(List<Course> courseList) {
        this.courseList = courseList;
    }
}

```

### 1.3.3映射文件

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

```

```

<mapper namespace="courseNamespace">

    <resultMap type="cn.itcast.javaee.mybatis.many2many.Course" id="courseMap">
        <id property="id" column="cid"/>
        <result property="name" column="cname"/>
    </resultMap>

    <!-- 查询哈哈选学了哪些课程 -->
    <select id="findAllByName" parameterType="string" resultMap="courseMap">
        select c.cid,c.cname
        from students s inner join middles m
        on s.sid = m.msid
        inner join courses c
        on m.mcid = c.cid
        and s.sname = #{name}
    </select>

</mapper>

```

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="studentNamespace">

    <resultMap type="cn.itcast.javaee.mybatis.many2many.Student"
id="studentMap">
        <id property="id" column="sid"/>
        <result property="name" column="sname"/>
    </resultMap>

    <select id="findAllByCourseName" parameterType="string"
resultMap="studentMap">
        select s.sname
        from students s inner join middles m
        on s.sid = m.msid
        inner join courses c
        on m.mcid = c.cid
        and c.cname = #{name}
    </select>

</mapper>

```

### 1.3.4DAO

```
package cn.itcast.javaee.mybatis.many2many;

import java.util.List;
import org.apache.ibatis.session.SqlSession;
import cn.itcast.javaee.mybatis.util.MybatisUtil;

/**
 * 持久层
 * @author AdminTC
 */
public class StudentCourseDao {
    /**
     * 查询哈哈选学了哪些课程
     * @param name 表示学生的姓名
     */
    public List<Course> findAllByName(String name) throws Exception{
        SqlSession sqlSession = null;
        try{
            sqlSession = MybatisUtil.getSqlSession();
            return sqlSession.selectList("courseNamespace.findAllByName",name);
        }catch(Exception e){
            e.printStackTrace();
            throw e;
        }finally{
            MybatisUtil.closeSqlSession();
        }
    }
    /**
     * 查询java课程有哪些学生选修
     * @param name 表示学生的课程
     */
    public List<Student> findAllByCourseName(String name) throws Exception{
        SqlSession sqlSession = null;
        try{
            sqlSession = MybatisUtil.getSqlSession();
            return
sqlSession.selectList("studentNamespace.findAllByCourseName",name);
        }catch(Exception e){
            e.printStackTrace();
            throw e;
        }finally{
            MybatisUtil.closeSqlSession();
        }
    }
}
```

```

}

public static void main(String[] args) throws Exception{
    StudentCourseDao dao = new StudentCourseDao();
    List<Course> courseList = dao.findAllByName("哈哈");
    System.out.print("哈哈选学了" + courseList.size()+"个课程,分别是: ");
    for(Course c : courseList){
        System.out.print(c.getName()+" ");
    }
    System.out.println("\n-----
-");
    List<Student> studentList = dao.findAllByCourseName("android");
    System.out.println("选修了android课程的学生有"+studentList.size()+"个, 分别是: ");
    for(Student s : studentList){
        System.out.print(s.getName()+" ");
    }
}
}

```

## 2. 关联映射总结

对于Mybatis的多表连接就非常简单了，由于SQL语句全是由我们自己写，如果我们返回的数据类型在当前的实体中是不够封装的话，那么我们只要再关联对应的映射属性就行了！



加油！加油！加油！



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

# 缓存+Mapper代理+逆向工程

---

## 1. 前言

---

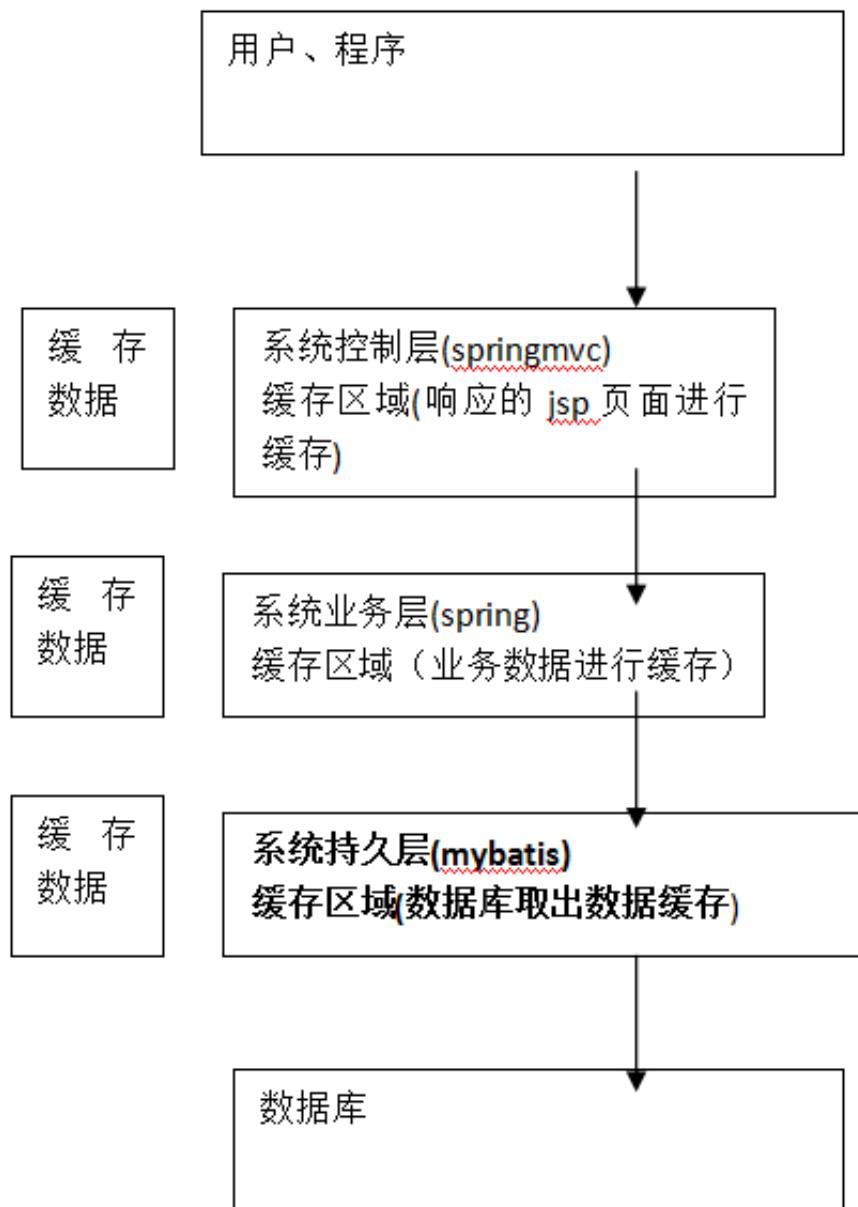
本文主要讲解Mybatis的以下知识点：

- Mybatis缓存
  - 一级缓存
  - 二级缓存
  - 与Ehcache整合
- Mapper代理
  - 使用Mapper代理就不用写实现类了
- 逆向工程
  - 自动生成代码

## 2. Mybatis缓存

---

缓存的意义将用户经常查询的数据放在缓存（内存）中，用户去查询数据就不用从磁盘上(关系型数据库数据文件)查询，从缓存中查询，从而提高查询效率，解决了高并发系统的性能问题。



[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

mybatis提供一级缓存和二级缓存



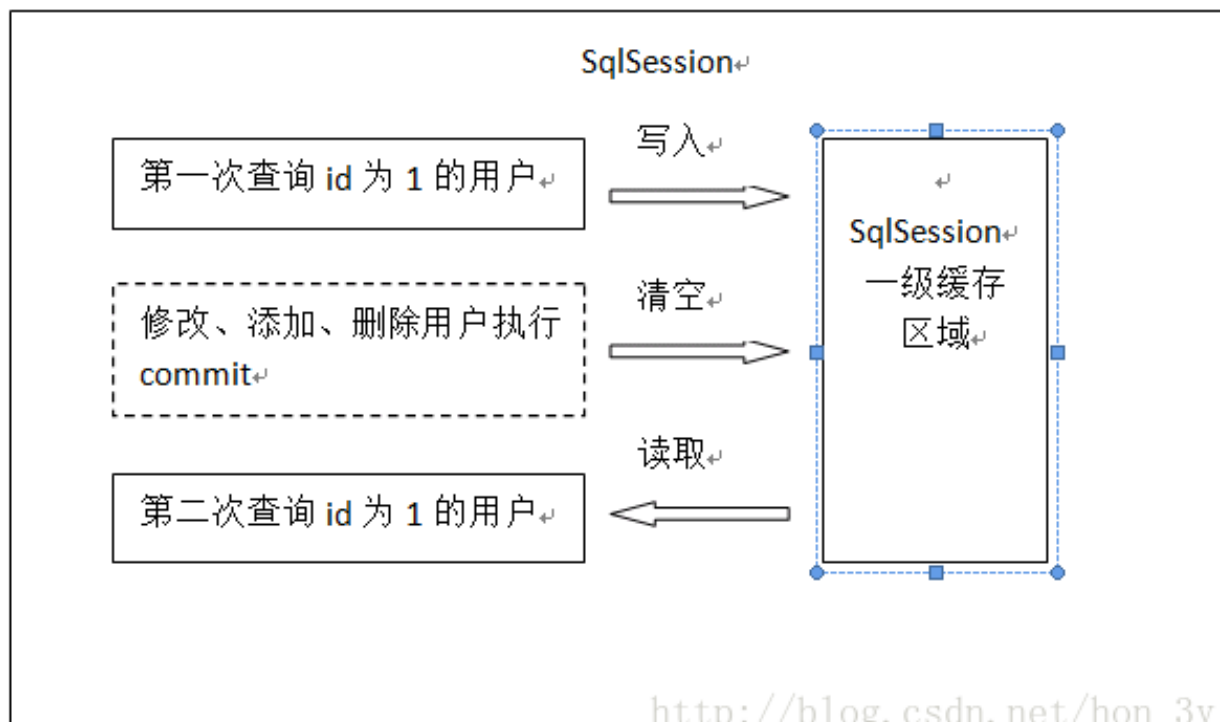
[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

- mybatis一级缓存是一个SqlSession级别，sqlsession只能访问自己的一级缓存的数据
- 二级缓存是跨sqlSession，是mapper级别的缓存，对于mapper级别的缓存不同的sqlsession是可以共享的。

看完上面对Mybatis的缓存的解释，我们发现Mybatis的缓存和Hibernate的缓存是极为相似的..

## 2.1 Mybatis一级缓存

Mybatis的一级缓存原理：



[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

第一次发出一个查询sql，sql查询结果写入sqlsession的一级缓存中，缓存使用的数据结构是一个map<key,value>

- key: hashCode+sql+sql输入参数+输出参数（sql的唯一标识）



- value: 用户信息

同一个sqlsession再次发出相同的sql, 就从缓存中取不走数据库。如果两次中间出现commit操作（修改、添加、删除），本sqlsession中的一级缓存区域全部清空，下次再去缓存中查询不到所以要从数据库查询，从数据库查询到再写入缓存。

每次查询都先从缓存中查询：

```
ry {
    queryStack++;
    list = resultHandler == null ? (List<E>) localCache.getObject(key) : null;
    if (list != null) {
```

如果缓存中查询到则将缓存数据直接返回。

如果缓存中查询不到就从数据库查询：

```
list = queryFromDatabase(ms, parameter, rowBounds, resultHandler, key, boundSql);
```

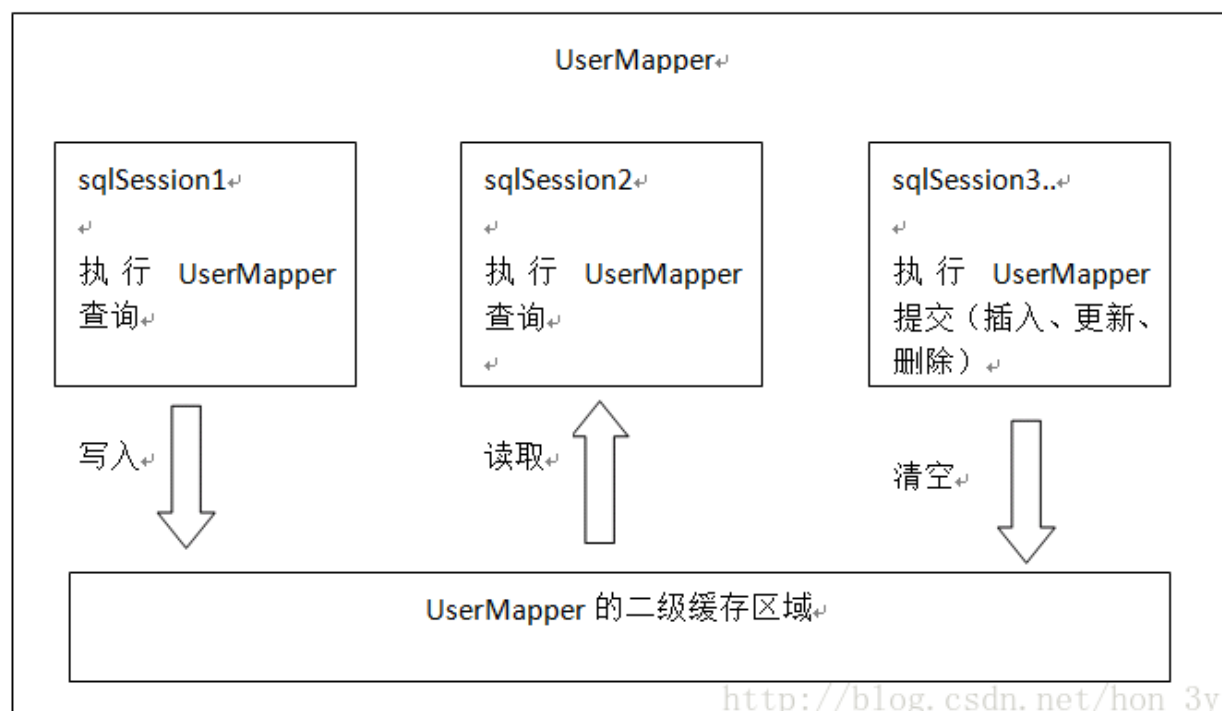
[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

Mybatis一级缓存值得注意的地方：

- Mybatis默认就是支持一级缓存的，并不需要我们配置。
- mybatis和spring整合后进行mapper代理开发，不支持一级缓存，mybatis和spring整合，spring按照mapper的模板去生成mapper代理对象，模板中在最后统一关闭sqlsession。

## 2.2 Mybatis二级缓存

二级缓存原理：



二级缓存的范围是mapper级别（mapper同一个命名空间），mapper以命名空间为单位创建缓存数据结构，结构是map<key、value>。

每次查询先看是否开启二级缓存，如果开启从二级缓存的数据结构中取缓存数据，

```
List<E> list = (List<E>) tcm.getObject(cache, key);
```

如果从二级缓存没有取到，再从一级缓存中找，如果一级缓存也没有，从数据库查询。|

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 2.3 Mybatis 二级缓存配置

需要我们在Mybatis的配置文件中配置二级缓存

```
<!-- 全局配置参数 -->
<settings>
  <!-- 开启二级缓存 -->
  <setting name="cacheEnabled" value="true"/>
</settings>
```

上面已经说了，二级缓存的范围是mapper级别的，因此我们的**Mapper**如果要使用二级缓存，还需要在对应的映射文件中配置..

```
<cache/>
```

## 2.4 查询结果映射的pojo序列化

mybatis二级缓存需要将查询结果映射的pojo实现 java.io.Serializable接口，如果不实现则抛出异常：

```
org.apache.ibatis.cache.CacheException: Error serializing object.
Cause: java.io.NotSerializableException: cn.itcast.mybatis.po.User
```

二级缓存可以将内存的数据写到磁盘，存在对象的序列化和反序列化，所以要实现java.io.Serializable接口。

如果结果映射的pojo中还包括了pojo，都要实现java.io.Serializable接口。

## 2.5 禁用二级缓存

对于变化频率较高的sql，需要禁用二级缓存：

在statement中设置useCache=false可以禁用当前select语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。、、

```
<select id="findOrderListResultMap" resultMap="ordersUserMap"
useCache="false">
```

## 2.6 刷新缓存

有的同学到这里可能会有一个疑问：为什么缓存我们都是在查询语句中配置？？而使用增删改的时候，缓存默认就会被清空【刷新了】？？？

缓存其实就是为我们的查询服务的，对于增删改而言，如果我们的缓存保存了增删改后的数据，那么再次读取时就会读到脏数据了！

我们在特定的情况下，还可以单独配置刷新缓存【但不建议使用】`flushCache`，默认是`true`

```
<update id="updateUser" parameterType="cn.itcast.mybatis.po.User"
flushCache="false">
    update user set username=#{username},birthday=#{birthday},sex=#{
sex},address=#{address} where id=#{id}
</update>
```

## 2.7 了解Mybatis缓存的一些参数

mybatis的cache参数只适用于mybatis维护缓存。

`flushInterval`（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

`size`（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

`readOnly`（只读）属性可以被设置为`true`或`false`。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是`false`。

如下例子：

```
<cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true"/>
```

这个更高级的配置创建了一个 `FIFO` 缓存，并每隔 60 秒刷新，存数结果对象或列表的 512 个引用，而且返回的对象被认为是只读的，因此在不同线程中的调用者之间修改它们会导致冲突。可用的回收策略有，默认的是 `LRU`：

1. `LRU` — 最近最少使用的：移除最长时间不被使用的对象。
2. `FIFO` — 先进先出：按对象进入缓存的顺序来移除它们。
3. `SOFT` — 软引用：移除基于垃圾回收器状态和软引用规则的对象。
4. `WEAK` — 弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象。



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

### 3.mybatis和ehcache缓存框架整合

---

ehcache是专门用于管理缓存的，Mybatis的缓存交由ehcache管理会更加得当..

在mybatis中提供一个cache接口，只要实现cache接口就可以把缓存数据灵活的管理起来。

```

public interface Cache {

    /**
     * @return The identifier of this cache
     */
    String getId();

    /**
     * @param key Can be any object but usually it is a {@link Cac
     * @param value The result of a select.
     */
    void putObject(Object key, Object value);

    /**
     * @param key The key
     * @return The object stored in the cache.
     */
    Object getObject(Object key);

    /**

```

mybatis 中默认实现：|

```

/**
 * @author Clinton Begin
 */
public class PerpetualCache implements Cache {

```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 3.1整合jar包

更新：如果项目是maven，用maven就好了

- mybatis-ehcache-1.0.2.jar
- ehcache-core-2.6.5.jar

ehcache对cache接口的实现类：

```

package org.mybatis.caches.ehcache;

import ...

public final class EhcacheCache implements Cache {
    private static final CacheManager CACHE_MANAGER = CacheManager.create();
    private final ReadWriteLock readWriteLock = new ReentrantReadWriteLock();
    private final String id;

    public EhcacheCache(String id) {
        if(id == null) {
            throw new IllegalArgumentException("Cache instances require an ID");
        } else {
            this.id = id;
            if(!CACHE_MANAGER.cacheExists(this.id)) {
                CACHE_MANAGER.addCache(this.id);
            }
        }
    }

    public void clear() { this.getCache().removeAll(); }

    public String getId() { return this.id; }

    public Object getObject(Object key) {
        try {

```

[http://blog.csdn.net/hon\\_3y](http://blog.csdn.net/hon_3y)

## 3.2 ehcache.xml配置信息

这个xml配置文件是配置全局的缓存管理方案

```

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">
    <!--diskStore: 缓存数据持久化的目录 地址 -->
    <diskStore path="F:\develop\ehcache" />
    <defaultCache
        maxElementsInMemory="1000"
        maxElementsOnDisk="10000000"
        eternal="false"
        overflowToDisk="false"
        diskPersistent="true"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU">
    </defaultCache>
</ehcache>

```

如果我们Mapper想单独拥有一些特性，需要在mapper.xml中单独配置

```
<!-- 单位：毫秒 -->
<cache type="org.mybatis.caches.ehcache.EhcacheCache">
  <property name="timeToIdleSeconds" value="12000"/>
  <property name="timeToLiveSeconds" value="3600"/>
  <!-- 同ehcache参数maxElementsInMemory -->
  <property name="maxEntriesLocalHeap" value="1000"/>
  <!-- 同ehcache参数maxElementsOnDisk -->
  <property name="maxEntriesLocalDisk" value="10000000"/>
  <property name="memoryStoreEvictionPolicy" value="LRU"/>
</cache>
```

## 3.3 应用场景与局限性

### 3.3.1 应用场景

对查询频率高，变化频率低的数据建议使用二级缓存。

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用mybatis二级缓存技术降低数据库访问量，提高访问速度

业务场景比如：

- 耗时较高的统计分析sql、
- 电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由mybatis每隔一段时间自动清空缓存，根据数据变化频率设置缓存刷新间隔flushInterval，比如设置为30分钟、60分钟、24小时等，根据需求而定。

### 3.3.2 局限性

#### mybatis局限性

mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用mybatis的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为mybaits的二级缓存区域以mapper为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此类问题需要在业务层根据需求对数据有针对性缓存。

## 4. Mapper代理方式

Mapper代理方式的意思就是：程序员只需要写dao接口，dao接口实现对象由mybatis自动生成代理对象。

经过我们上面的几篇博文，我们可以发现我们的DaoImpl是十分重复的...

1 dao的实现类中存在重复代码，整个mybatis操作的过程代码模板重复（先创建sqlsession、调用sqlsession的方法、关闭sqlsession）

2、dao的实现 类中存在硬编码，调用sqlsession方法时将statement的id硬编码。

以前的重复代码和硬编码如下：

```

public class StudentDao {

    public void add(Student student) throws Exception {
        //得到连接对象
        SqlSession sqlSession = MybatisUtil.getSqlSession();
        try{
            //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL
            sqlSession.insert("StudentID.add", student);
            sqlSession.commit();
        }catch(Exception e){
            e.printStackTrace();
            sqlSession.rollback();
            throw e;
        }finally{
            MybatisUtil.closeSqlSession();
        }
    }

    public static void main(String[] args) throws Exception {
        StudentDao studentDao = new StudentDao();
        Student student = new Student(3, "zhong3", 10000D);
        studentDao.add(student);
    }
}

```

## 4.1 Mapper开发规范

想要Mybatis帮我们自动生成Mapper代理的话, 我们需要遵循以下的规范:

- 1、**mapper.xml**中**namespace**指定为**mapper**接口的全限定名
  - 此步骤目的: 通过**mapper.xml**和**mapper.java**进行关联。
- 2、**mapper.xml**中**statement**的**id**就是**mapper.java**中方法名
- 3、**mapper.xml**中**statement**的**parameterType**和**mapper.java**中方法输入参数类型一致
- 4、**mapper.xml**中**statement**的**resultType**和**mapper.java**中方法返回值类型一致。

再次说明: **statement**就是我们在**mapper.xml**文件中命名空间+sql指定的id

## 4.2 Mapper代理返回值问题

mapper接口方法返回值:

- 如果是返回的单个对象, 返回值类型是pojo类型, 生成的代理对象内部通过selectOne获取记录
- 如果返回值类型是集合对象, 生成的代理对象内部通过selectList获取记录。

## 5. Mybatis解决JDBC编程的问题

- 1、数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能, 如果使用数据库链接池可解决此问题。



- 解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

2、Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

- 解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

3、向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

- 解决：Mybatis自动将java对象映射至sql语句，通过statement中的parameterType定义输入参数的类型。

4、对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

- 解决：Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

## 6.Mybatis逆向工程

Mybatis逆向工程实际上就是想自动生成对应的代码，不用自己写对应的映射文件和接口（又偷懒了）

借鉴博文：[http://blog.csdn.net/for\\_my\\_life/article/details/51228098](http://blog.csdn.net/for_my_life/article/details/51228098)

### 6.1 修改pom.xml文件

向该工程添加逆向工程插件..

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>asdf</groupId>
    <artifactId>asdf</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <finalName>zhongfucheng</finalName>
        <plugins>
            <plugin>
                <groupId>org.mybatis.generator</groupId>
                <artifactId>mybatis-generator-maven-plugin</artifactId>
                <version>1.3.2</version>
                <configuration>
                    <verbose>true</verbose>
                    <overwrite>true</overwrite>
```

```

        </configuration>
    </plugin>
</plugins>
</build>

</project>

```

## 6.2 generatorConfig.xml 配置文件

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
    PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
    "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<generatorConfiguration>
    <!--
        <properties resource="conn.properties" />
        -->
    <!-- 处理1, 这里的jar包位置可能需要修改 -->
    <classPathEntry location="C:\mybatisMaven\lib\mysql-connector-java-5.1.7-
bin.jar"/>
    <!-- 指定运行环境是mybatis3的版本 -->
    <context id="testTables" targetRuntime="MyBatis3">

        <commentGenerator>
            <!-- 是否取消注释 -->
            <property name="suppressAllComments" value="true" />
            <!-- 是否生成注释代时间戳 -->
            <property name="suppressDate" value="true" />
        </commentGenerator>
        <!-- 处理2    jdbc 连接信息, 看看库是否存在 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
            connectionURL="jdbc:mysql://localhost:3306/scm?
useUnicode=true&characterEncoding=UTF-8" userId="root" password="root">
        </jdbcConnection>

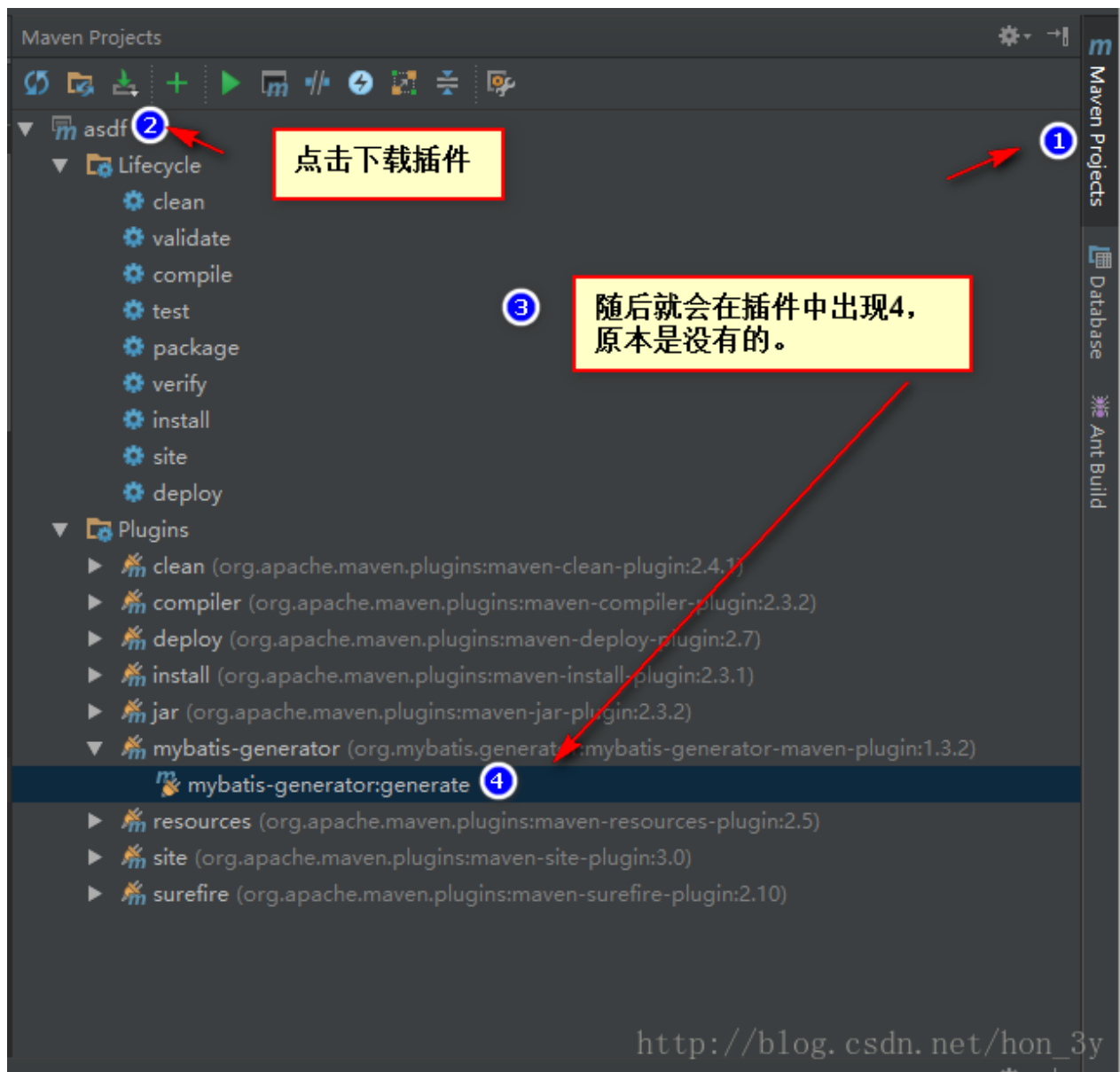
        <!--处理3    targetPackage指定模型在生成在哪个包 ,targetProject指定项目的
src,-->
        <javaModelGenerator targetPackage="zhongfucheng.entity"
            targetProject="src/main/java">
            <!-- 去除字段前后空格 -->
            <property name="trimStrings" value="false" />
        </javaModelGenerator>
        <!--处理4    配置SQL映射文件生成信息 -->
        <sqlMapGenerator targetPackage="zhongfucheng.dao"
            targetProject="src/main/java" />
        <!-- 处理5    配置dao接口生成信息-->
    </context>

```

```
<javaClientGenerator type="XMLMAPPER" targetPackage="zhongfucheng.dao"
targetProject="src/main/java" />

<table tableName="account" domainObjectName="Account"/>
<table tableName="supplier" domainObjectName="Supplier"/>
</context>
</generatorConfiguration>
```

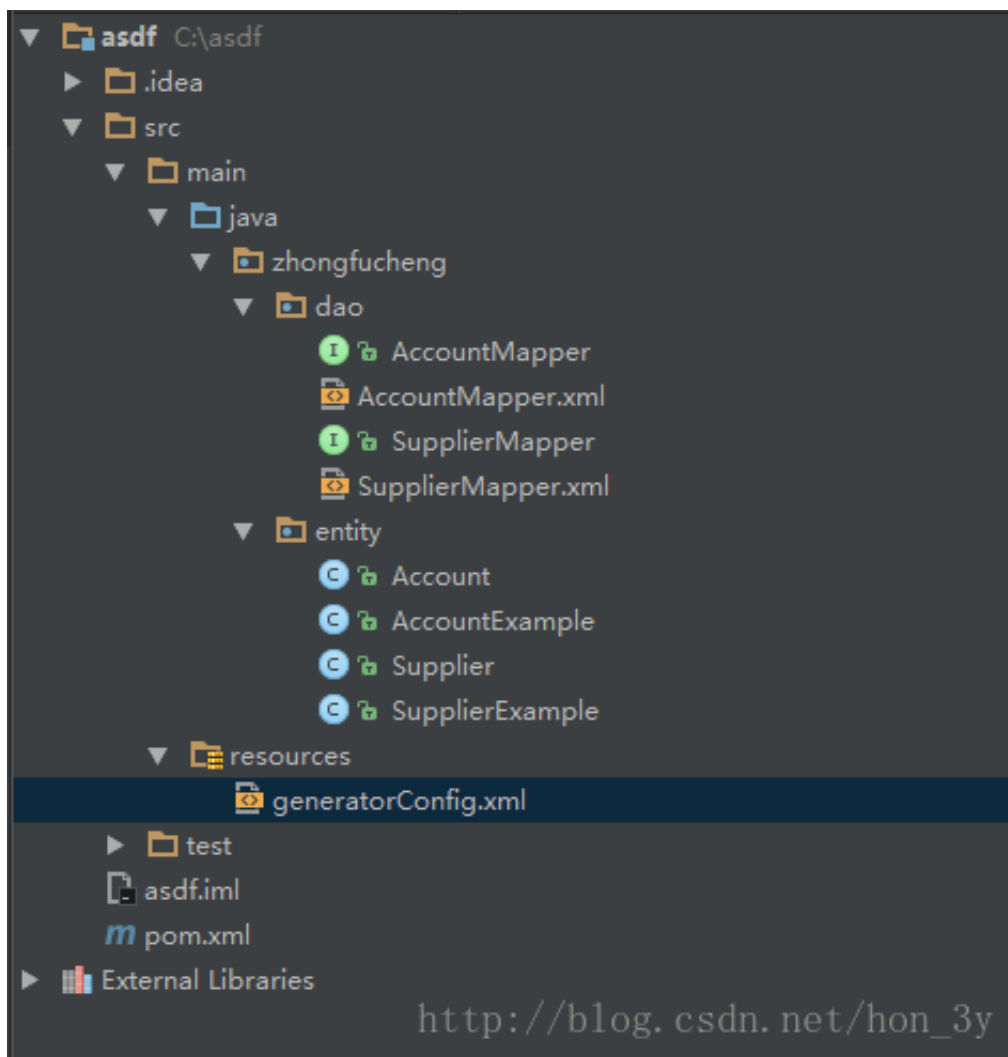
## 6.3 使用插件步骤



## 6.4 最后生成代码

如果对我们上面generatorConfig.xml配置的包信息不清楚的话，那么可以看一下我们的完整项目结构图...

因为我们在Idea下是不用写对应的工程名字的，而在eclipse是有工程名字的。



## 7.本章总结

- Mybatis的一级缓存是sqlSession级别的。只能访问自己的sqlSession内的缓存。如果Mybatis与Spring整合了，Spring会自动关闭sqlSession的。所以一级缓存会失效的。
- 一级缓存的原理是map集合，Mybatis默认就支持一级缓存
- 二级缓存是Mapper级别的。只要在Mapper命名空间下都可以使用二级缓存。需要我们自己去配置二级缓存
- Mybatis的缓存我们可以使用Ehcache框架来进行管理，Ehcache实现Cache接口就代表使用Ehcache来环境Mybatis缓存。
- 由于之前写的DaoImpl是有非常多的硬编码的。可以使用Mapper代理的方式来简化开发
  - 命名空间要与JavaBean的全类名相同
  - sql片段语句的id要与Dao接口的方法名相同
  - 方法的参数和返回值要与SQL片段的接收参数类型和返回类型相同。



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

# Mybatis整合Spring

---

## 1. Mybatis与Spring整合

---

既然我们已经学了Mybatis的基本开发了，接下来就是Mybatis与Spring的整合了！

以下使用的是Oracle数据库来进行测试

### 1.1 导入jar包

更新：如果用Maven的话，直接使用Maven就好了

- aopalliance.jar
- asm-3.3.1.jar

- aspectjweaver.jar
- c3p0-0.9.1.2.jar
- cglib-2.2.2.jar
- commons-logging.jar
- log4j-1.2.16.jar
- mybatis-3.1.1.jar
- mybatis-spring-1.1.1.jar
- mysql-connector-java-5.1.7-bin.jar
- ojdbc5.jar
- org.springframework.aop-3.0.5.RELEASE.jar
- org.springframework.asm-3.0.5.RELEASE.jar
- org.springframework.beans-3.0.5.RELEASE.jar
- org.springframework.context-3.0.5.RELEASE.jar
- org.springframework.core-3.0.5.RELEASE.jar
- org.springframework.expression-3.0.5.RELEASE.jar
- org.springframework.jdbc-3.0.5.RELEASE.jar
- org.springframework.orm-3.0.5.RELEASE.jar
- org.springframework.transaction-3.0.5.RELEASE.jar
- org.springframework.web.servlet-3.0.5.RELEASE.jar
- org.springframework.web-3.0.5.RELEASE.jar

## 1.2 创建表

```
create table emps(  
    eid number(5) primary key,  
    ename varchar2(20),  
    esal number(8,2),  
    esex varchar2(2)  
);
```

## 1.3 创建实体

```
package entity;  
  
/**  
 * 员工  
 * @author AdminTC  
 */  
public class Emp {  
    private Integer id;  
    private String name;  
    private Double sal;  
    private String sex;  
    public Emp() {}  
    public Emp(Integer id, String name, Double sal, String sex) {
```

```

        this.id = id;
        this.name = name;
        this.sal = sal;
        this.sex = sex;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Double getSal() {
        return sal;
    }
    public void setSal(Double sal) {
        this.sal = sal;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
}

```

## 1.4 创建实体与表的映射文件

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="empNamespace">

    <resultMap type="entity.Emp" id="empMap">
        <id property="id" column="eid"/>
        <result property="name" column="ename"/>
        <result property="sal" column="esal"/>
        <result property="sex" column="esex"/>
    </resultMap>

    <!-- 增加员工 -->

```

```

<insert id="add" parameterType="entity.Emp">
    insert into emps(eid,ename,esal,esex) values(#{id},#{name},#{sal},#{sex})
</insert>

</mapper>

```

## 1.5 创建Mybatis映射文件配置环境

数据库的信息交由Spring管理！Mybatis配置文件负责加载对应映射文件即可

```

<mappers>
    <mapper resource="zhongfucheng/entity/EmpMapper.xml"/>

</mappers>
</configuration>

```

## 1.6 配置Spring核心过滤器【也是加载总配置文件】

```

<!-- 核心springmvc核心控制器 -->
<servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring.xml</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>

```

## 1.7 配置数据库信息、事务

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"

```



```

        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 配置C3P0连接池,目的: 管理数据库连接 -->
    <bean id="comboPooledDataSourceID"
class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="jdbcUrl"
value="jdbc:oracle:thin:@127.0.0.1:1521:ZHONGFUCHENG"/>
        <property name="user" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>

    <!-- 配置SqlSessionFactoryBean, 目的: 加载mybaitis配置文件和映射文件, 即替代原
Mybatis工具类的作用 -->
    <bean id="sqlSessionFactoryBeanID"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="configLocation" value="classpath:mybatis.xml"/>
        <property name="dataSource" ref="comboPooledDataSourceID"/>
    </bean>

    <!-- 配置Mybatis的事务管理器, 即因为Mybatis底层用的是JDBC事务管理器, 所以在这里依然配
置JDBC事务管理器 -->
    <bean id="dataSourceTransactionManagerID"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="comboPooledDataSourceID"/>
    </bean>

    <!-- 配置事务通知, 即让哪些方法需要事务支持 -->
    <tx:advice id="tx" transaction-manager="dataSourceTransactionManagerID">
        <tx:attributes>
            <tx:method name="*" propagation="REQUIRED"/>
        </tx:attributes>
    </tx:advice>

    <!-- 配置事务切面, 即让哪些包下的类需要事务 -->
    <aop:config>
        <aop:pointcut id="pointcut" expression="execution(*
zhongfucheng.service.*(..))"/>
        <aop:advisor advice-ref="tx" pointcut-ref="pointcut"/>
    </aop:config>

```

```
</aop:config>

<!--扫描注解-->
<context:component-scan base-package="zhongfucheng"/>

</beans>
```

## 1.8 创建Dao、Service、Action

```
@Repository
public class EmpDao {
    @Autowired
    private SqlSessionFactory sqlSessionFactory;
    /**
     * 增加员工
     */
    public void add(Emp emp) throws Exception {
        SqlSession sqlSession = sqlSessionFactory.openSession();
        sqlSession.insert("empNamespace.add", emp);
        sqlSession.close();
    }
}
```

```
@Service
public class EmpService {

    @Autowired
    private zhongfucheng.dao.EmpDao empDao;
    public void addEmp(Emp emp) throws Exception {
        empDao.add(emp);
    }
}
```

```
@Controller
@RequestMapping("/emp")
public class EmpAction {

    @Autowired
    private EmpService empService;
```

```

    @RequestMapping("/register")
    public void register(Emp emp) throws Exception {
        empService.addEmp(emp);
        System.out.println("注册成功");
    }
}

```

## 1.9JSP页面测试

```

<%@ page language="java" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>员工注册</title>
    </head>
    <body>
        <form action="{pageContext.request.contextPath}/emp/register.action"
method="POST">
            <table border="2" align="center">
                <tr>
                    <th>编号</th>
                    <td><input type="text" name="id"></td>
                </tr>
                <tr>
                    <th>姓名</th>
                    <td><input type="text" name="name"></td>
                </tr>
                <tr>
                    <th>薪水</th>
                    <td><input type="text" name="sal"></td>
                </tr>
                <tr>
                    <th>性别</th>
                    <td>
                        <input type="radio" name="sex" value="男"/>男
                        <input type="radio" name="sex" value="女" checked/>女
                    </td>
                </tr>
                <tr>
                    <td colspan="2" align="center">
                        <input type="submit" value="注册"/>
                    </td>
                </tr>
            </table>
        </form>
    </body>

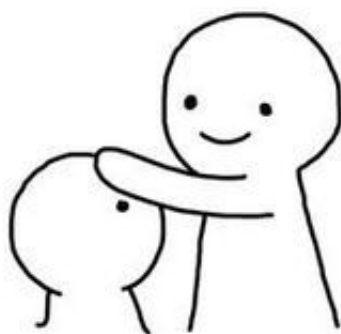
```

```
</html>
```

## 2. 总结

---

- web.xml加载Spring配置文件
- Spring配置文件配置数据连接池，SessionFactory、事务、扫描注解
- Mybatis总配置文件、实体以及相对应的映射文件
- 将映射文件加入到总配置文件中。



加油加油



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多原创技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>

## Mybatis常见面试题

---

### 1. `#{}和${}`的区别是什么？

---

`#{}和${}`的区别是什么？

在Mybatis中，有两种占位符

- `#{}解析`传递进来的参数数据
- `${}`对传递进来的参数原样拼接在SQL中
- `#{}是预编译处理`，`${}`是字符串替换。
- 使用`#{}可以有效的防止SQL注入`，提高系统安全性。

## 2.当实体类中的属性名和表中的字段名不一样，怎么办？

当实体类中的属性名和表中的字段名不一样，怎么办？

第1种：通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致

```
<select id="selectorder" parametertype="int"
resulttype="me.gacl.domain.order">
    select order_id id, order_no orderno ,order_price price form orders
where order_id=#{id};
</select>
```

第2种：通过 `<resultMap>` 来映射字段名和实体类属性名的一一对应的关系

```
<select id="getOrder" parameterType="int" resultMap="orderresultmap">
    select * from orders where order_id=#{id}
</select>
<resultMap type="me.gacl.domain.order" id="orderresultmap">
    <!--用id属性来映射主键字段-->
    <id property="id" column="order_id">
    <!--用result属性来映射非主键字段，property为实体类属性名，column为数据表中的属性-->
    <result property = "orderno" column ="order_no"/>
    <result property="price" column="order_price" />
</resultMap>
```

我认为第二种方式会好一点。

## 3. 如何获取自动生成的(主)键值？

如何获取自动生成的(主)键值？

如果我们一般插入数据的话，如果我们想要知道刚刚插入的数据的主键是多少，我们可以通过以下的方式来获取

需求：user对象插入到数据库后，新记录的主键要通过user对象返回，通过user获取主键值。

解决思路：通过LAST\_INSERT\_ID()获取刚插入记录的自增主键值，在insert语句执行后，执行select LAST\_INSERT\_ID()就可以获取自增主键。

mysql:

```
<insert id="insertUser" parameterType="cn.itcast.mybatis.po.User">
  <selectKey keyProperty="id" order="AFTER" resultType="int">
    select LAST_INSERT_ID()
  </selectKey>
  INSERT INTO USER(username,birthday,sex,address) VALUES(#{username},#{
birthday},#{sex},#{address})
</insert>
```

oracle:实现思路：先查询序列得到主键，将主键设置到user对象中，将user对象插入数据库。

```
<!-- oracle
在执行insert之前执行select 序列.nextval() from dual取出序列最大值，将值设置到user对
象 的id属性
-->
<insert id="insertUser" parameterType="cn.itcast.mybatis.po.User">
  <selectKey keyProperty="id" order="BEFORE" resultType="int">
    select 序列.nextval() from dual
  </selectKey>

  INSERT INTO USER(id,username,birthday,sex,address) VALUES( 序列.nextval(),#{
username},#{birthday},#{sex},#{address})
</insert>
```

也可以在 select 标签下写以下的属性：

```
< select useGeneratedKeys="true" keyProperty="id" keyColumn="id" />
```

## 4. 在mapper中如何传递多个参数？

在mapper中如何传递多个参数？

第一种：使用占位符的思想

- 在映射文件中使用#{0},#{1}代表传递进来的第几个参数
- 使用@param注解:来命名参数
- #{0},#{1} 方式

//对应的xml,#{0}代表接收的是dao层中的第一个参数,#{1}代表dao层中第二参数,更多参数一致往后加即可。

```
<select id="selectUser" resultMap="BaseResultMap">
    select * from user_user_t where user_name = #{0} and user_area = #{1}
</select>
```

- @param注解方式

```
public interface usermapper {
    user selectuser(@param("username") string username,
        @param("hashedpassword") string hashedpassword);
}
```

```
<select id="selectuser" resultType="user">
    select id, username, hashedpassword
    from some_table
    where username = #{username}
    and hashedpassword = #{hashedpassword}
</select>
```

## 第二种：使用Map集合作为参数来装载

```
try{
    //映射文件的命名空间.SQL片段的ID, 就可以调用对应的映射文件中的SQL

    /**
     * 由于我们的参数超过了两个, 而方法中只有一个Object参数收集
     * 因此我们使用Map集合来装载我们的参数
     */
    Map<String, Object> map = new HashMap();
    map.put("start", start);
    map.put("end", end);
    return sqlSession.selectList("StudentID.pagination", map);
}catch(Exception e){
    e.printStackTrace();
    sqlSession.rollback();
    throw e;
}finally{
    MybatisUtil.closeSqlSession();
}
```

```
<!--分页查询-->
<select id="pagination" parameterType="map" resultMap="studentMap">

    /*根据key自动找到对应Map集合的value*/
    select * from students limit #{start},#{end};

</select>
```

## 5. Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？

Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？

- Mybatis动态sql可以让我们在Xml映射文件内，以标签的形式编写动态sql，完成逻辑判断和动态拼接sql的功能。
- Mybatis提供了9种动态sql标签：trim|where|set|foreach|if|choose|when|otherwise|bind。
- 其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能。

## 6. Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

如果配置了namespace那么当然是可以重复的，因为我们的Statement实际上就是namespace+id

如果没有配置namespace的话，那么相同的id就会导致覆盖了。

## 7. 为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

- Hibernate属于全自动ORM映射工具，使用Hibernate查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。
- 而Mybatis在查询关联对象或关联集合对象时，需要手动编写sql来完成，所以，称之为半自动ORM映射工具。

## 8. 通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？



通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

- Dao接口，就是人们常说的Mapper接口，接口的全限定名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。
- Mapper接口是没有实现类的，当调用接口方法时，接口全限定名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement

举例：

```
com.mybatis3.mappers.StudentDao.findStudentById,
```

可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MappedStatement对象。

Dao接口里的方法，是不能重载的，因为是全限定名+方法名的保存和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

详情可参考：

- <https://www.cnblogs.com/soundcode/p/6497291.html>

## 9. Mybatis比IBatis比较大的几个改进是什么

Mybatis比IBatis比较大的几个改进是什么

- a.有接口绑定,包括注解绑定sql和xml绑定Sql,
- b.动态sql由原来的节点配置变成OGNL表达式,
- c. 在一对一,一对多的时候引进了association,在一对多的时候引入了collection节点,不过都是在resultMap里面配置

## 10. 接口绑定有几种实现方式,分别是怎么实现的?

接口绑定有几种实现方式,分别是怎么实现的?

接口绑定有两种实现方式：

- 一种是通过注解绑定,就是在接口的方法上面加上@Select@Update等注解里面包含Sql语句来绑定
- 另外一种就是通过xml里面写SQL来绑定,在这种情况下,要指定xml映射文件里面的namespace必须为接口的全路径名.

## 11. Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

举例：`select * from student`，拦截sql后重写为：`select t.* from (select * from student) t limit 0, 10`

分页插件使用参考资料：

- <https://www.cnblogs.com/kangoroo/p/7998433.html>
- <http://blog.csdn.net/yuchao2015/article/details/55001182>
- <https://www.cnblogs.com/ljdblog/p/6725094.html>

## 12. 简述Mybatis的插件运行原理，以及如何编写一个插件

简述Mybatis的插件运行原理，以及如何编写一个插件

Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

## 13. Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载 `lazyLoadingEnabled=true|false`。

它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 `a.getB().getName()`，拦截器invoke()方法发现 `a.getB()` 是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用 `a.setB(b)`，于是a的对象b属性就有值了，接着完成 `a.getB().getName()` 方法的调用。这就是延迟加载的基本原理。

当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的。

## 14. Mybatis都有哪些Executor执行器？它们之间的区别是什么？

Mybatis都有哪些Executor执行器？它们之间的区别是什么？

Mybatis有三种基本的Executor执行器，**SimpleExecutor**、**ReuseExecutor**、**BatchExecutor**。

- SimpleExecutor：每执行一次update或select，就开启一个Statement对象，**用完立刻关闭Statement对象**。
- ReuseExecutor：执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map<String, Statement>内，供下一次使用。简言之，**就是重复使用Statement对象**。
- BatchExecutor：执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），等待统一执行（executeBatch()），**它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，等待逐一执行executeBatch()批处理。与JDBC批处理相同**。

作用范围：Executor的这些特点，都严格限制在SqlSession生命周期范围内。

## 15. MyBatis与Hibernate有哪些不同？

### MyBatis与Hibernate有哪些不同？

Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句，不过mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。

Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一旦需求变化要求成果输出迅速。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。

Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的缺点是学习门槛高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。

总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。



加油加油



如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：<https://github.com/ZhongFuCheng3y/3y>